# EMOTION AND SENTIMENT ANALYSES WITH GAUSSIAN PROCESS MULTI-TASK KERNELS

A Thesis

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Mathematics

By

Christopher Muzquiz

2022

## SIGNATURE PAGE

| | |
|---|---|
| **THESIS:** | EMOTION AND SENTIMENT ANALYSES WITH GAUSSIAN PROCESS MULTI-TASK KERNELS |
| **AUTHOR:** | Christopher Muzquiz |
| **DATE SUBMITTED:** | Summer 2022 |
| | Department of Mathematics and Statistics |

Dr. Jimmy Risk _____
Thesis Committee Chair
Mathematics & Statistics

Dr. Adam King _____
Mathematics & Statistics

Dr. Michael Green _____
Mathematics & Statistics

## ACKNOWLEDGMENTS

Completing a master's degree is no easy feat, and this would not have been possible if it was not for my support systems whom I appreciate so much.

I want to thank the people that supported me during the journey of completing my master's degree and continue to support me in life. This is directed toward my beloved family and friends whom endlessly help and encourage me. My loved ones are amazing, wonderful people, and I am so grateful to have them in my life. A big thanks goes to my brother, Mark, for running scripts on his computer for this project.

I also want to give a special thanks to my thesis advisor, Dr. Jimmy Risk. Dr. Risk has been immensely resourceful, insightful, supportive, and an awesome mentor. I have a lot of respect for Dr. Risk for he is such a brilliant professor, and I am so grateful he was willing to take me under his wing. He really is one of my favorite people to talk to because we engage in such interesting conversations about mathematics and statistics.

Finally, I want to give thanks to the Dr. Adam King and Dr. Michael Green whom agreed to act as thesis committee members. Dr. King and Dr. Green are both great professors, and I know this because I have taken statistics courses with both of them in the past. I am honored that they were willing to be a part of my thesis committee, and I appreciate their time and any advice they had to give.

## ABSTRACT

Natural language processing (NLP) systems analyze language to extract linguistic information and build representations from data. The tasks of emotion and sentiment analyses are among many currently being studied and will be addressed in this thesis. The first half of this paper is the theoretical development of Gaussian processes, kernel functions, multi-task learning, string processing in NLP, and methods of model assessment and selection. The objective of the latter half is to implement Gaussian process models to predict scores of affect on a dataset containing annotated headlines. We construct single-task models equipped with the string, polynomial, and Matérn kernels and use compound kernels produced by means of forward stepwise selection. We then build multi-task models and employ kernels obtained under the linear model of coregionalization assumption. Lastly, we analyze the performance of our models using cross-validation and test set prediction assessment metrics.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Natural language processing (NLP) is the theoretically motivated computational field concerned with analyzing and representing linguistic information for the purpose of addressing tasks that require language understanding [38]. NLP is considered to be a facet of artificial intelligence since one of its goals is to imitate human performance [38].

Psycholinguists study the *synchronic* model of language which posits that levels of language processing occur dynamically, rather than sequentially. The levels of language consist of the following [38]:

- phonology, the organization of speech and sounds,
- morphology, the composure of morphemes (the smallest units of meaning),
- lexical, the interpretation of meanings of words,
- syntactic, the parsing of grammatical structure of sentences,
- semantic, the interactions among word-level meanings in a sentence,

- discourse, the properties of text across multiple sentences, and
- pragmatic, the use of context to guide understanding of words.

The three different learning approaches to NLP are the symbolic, statistical, and connectionist approaches, as described by Wermter et al. [66]. The symbolic approach employs inductive learning algorithms to develop general rules by observation of examples. The statistical approach uses large text corpora and analyzes text characteristics without adding linguistic or world knowledge. The connectionist approach is the modeling of language learning systems by means of artificial neural networks.

Developments in NLP lead to the emergence of new technologies that shape our daily lives. Google Translate serves as a primary example of a machine translator in which a user can input text in one language, and by means of NLP, the system outputs the text in a different language. Other tasks in NLP include information retrieval [43, 35], information extraction [32], question-answering [1], summarization [22], and dialogue systems [13], to name a few.

In addition to the tasks previously mentioned, another pair of important tasks are emotion and sentiment analyses. These are the tasks that we will address in this thesis. We will be examining the *SemEval 2007 Affective Text Dataset* [59] (Affective Text), a dataset commonly studied by others for the emotion analysis task [6, 58, 7]. This dataset contains news headlines with associated annotated emotion and valence scores. The score system in this dataset is founded on the theory of expressive emotional families by Ekman [18] and the dimensional models of emotion proposed by Schlosberg [55] and Russell [53].

Ekman [19] argued that emotional responses are unbidden (involuntary) physiological reactions and recognized that emotions constitute distinct families of af-

fective states across cultures [18]. He reported that there are universal facial expressions that can be distinguished as anger, fear, enjoyment, sadness, and disgust. Ekman [18] stated that "Ectoff and Magee (in press) found evidence that surprise is perceived differently than other emotions, not defining an exclusive category."

In the dimensional models of emotional states proposed by Schlosberg [55] and Russell [53], the common factor between these models is the pleasant-unpleasant dimension. This acts as a hedonic measure and is another way of describing *sentiment* or *valence* [5]. In psychology, a positive (or negative) valence refers to the intrinsic attractiveness (or aversiveness) that events, objects, or situations possess [21].

Emotion and sentiment analyses are technically distinct but fundamentally related tasks. *Emotion analysis* is a part of affective computing that relates to the extraction of information to infer emotions from natural language data [25]. *Sentiment analysis* is the extraction of information from natural language to detect a favorable or unfavorable opinion about a subject matter or topic [44], though the task at hand is technically a type of opinion polarity classification [69].

Research is active when it comes to emotion analysis. For example, Xia and Ding [67] formulated a new task which they call *emotion-cause pair extraction* as an extension to the task of *emotion cause extraction*. Buechel and Hahn [10] studied the impact of annotation perspective and representations in text corpora and experimented with the mapping of emotions between dimensional and categorical formats. Relating to the dataset that we will be using, Staiano and Guerini [58] collected news articles from a news site along with data produced by a function within the site for the reader to report the emotion that was evoked upon reading an article, allowing for the creation of an emotion lexicon with text data and

crowd-sourced annotations. They then tested their model using the Affective Text dataset to predict emotion scores. Beck [6] has done an analysis of Affective Text dataset while also using Gaussian processes. The work done here is similar but distinguishable because we are using the test set as a validation set rather than combining the development and test sets to perform $k$-fold cross-validation, and thus we are training with less data; there is a focus on predicting valence scores for the sentiment analysis task, adding another layer to our analysis; we introduce compound kernels through forward stepwise selection; and multi-task Guassian process models will be implemented that use kernels obtained under the linear model of coregionalization assumption.

In this thesis, we will also cover the string subsequence kernel, a kernel approach to processing strings of text originally formulated by Lodhi et al. [39]. Lodhi et al. [40] later explored different parameterizations of the string subsequence kernel, discussed the kernel's effectiveness in experimental settings, and introduced an approximation method. Cancedda et al. [11] then introduced the word sequence kernel, a variation of the string subsequence kernel that operates at the word level, and introduced new hyperparameters. Finally, Beck [6] introduced a vectorized approach of the word sequence kernel, which we use in our Gaussian process models.

In Chapter 2, we provide the theoretical foundations for Gaussian processes, explaining the model from weight-space and function-space views. Next in Chapter 3, we provide the necessary background to use kernels with Gaussian processes. We introduce common covariance (kernel) functions, and we set the stage so that we can compound kernels. Chapter 4 then introduces multi-task Gaussian processes and linear models of coregionalization to allow for multi-task learning in our models. In Chapter 5, we discuss strings in the context of NLP. We introduce hard

matching and soft matching embedding approaches, and we introduce the string kernel. Chapter 6 lays out the ground work for model assessment and selection with the discussion of error metrics, resampling methods, and forward stepwise selection. Chapter 7 is the empirical data analysis of the Affective Text dataset [59]. This chapter discusses the data, the preprocessing procedure, the embedding procedure, and custom kernels. Then, we proceed to conduct emotion and sentiment analyses with Gaussian processes. These are done with independent single-task and multi-task models. Finally, we close out the thesis by providing some discussion of our data analysis and mention prospects of future work.

# Chapter 2

# Gaussian Processes

A Gaussian process (GP) is a non-parametric linear model that can be used to address regression and classification problems in a supervised learning fashion. It is a stochastic process that can be defined in a weight-space view, as a distribution over a parameterized function, or an equivalent function-space view, as a set of random variables that share a distribution. The majority of the information in this chapter about Gaussian process regression (GPR) is a summarization of the theoretical foundations laid out in Gaussian Processes for Machine Learning (2005) by Rasmussen and Williams [49, Ch. 2].

The notation that we will use to denote the training set of $N$ observations is $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, N\}$, where $\mathbf{x}$ denotes the $D$-dimensional input vector and $y$ denotes the scalar output. This implies that our aggregated input matrix of covariates $X$, known as the *design matrix*, has shape $D \times N$. In regression, the target values are stored in the vector $\mathbf{y} = (y_1, \ldots, y_N)^\top$, so we may write $\mathcal{D} = (X, \mathbf{y})$. We will also use the notation of a subscripted asterisk (*) to represent data from the test set of size $N_*$. That is, $X_*$ is the $D \times N_*$ matrix of test inputs.

## 2.1 Weight-space View

Linear models are enhanced where we project the inputs into a high-dimensional feature space and apply the linear model there. In some feature spaces when the dimensionality of the feature space is large compared to the number of data points, we can take advantage of the *kernel trick* in which we implicitly carry out the computations in the high-dimensional space.

### 2.1.1 Motivation with Standard Linear regression

A standard linear regression model with Gaussian noise takes the form

$$y = f(\mathbf{x}) + \varepsilon = \mathbf{x}^\top \mathbf{w} + \varepsilon \tag{2.1}$$

such that $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$. The vector $\mathbf{x}$ is generally augmented by having an additional value of one to account for a bias weight or offset. For a set of observations $\mathbf{y}$, we assume that the additive noise is independent identically distributed (iid) Gaussian with zero mean and variance $\sigma_\varepsilon^2$, which we denote

$$\varepsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\varepsilon^2). \tag{2.2}$$

The model with this noise assumption give us the notion of a likelihood. Given the parameters, a *likelihood* is the probability density of the observations that is factored over cases in the training set as permitted by the independence assumption. Since $X$ is non-random, we have that

$$\begin{aligned} p(\mathbf{y}|\mathbf{w}) &= \prod_{i=1}^{N} p(y_i|\mathbf{w}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_\varepsilon} \exp\left( - \frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2\sigma_\varepsilon^2} \right) \\ &= \frac{1}{(2\pi\sigma_\varepsilon^2)^{N/2}} \exp\left( - \frac{1}{2\sigma_\varepsilon^2} |\mathbf{y} - X^\top \mathbf{w}|^2 \right) = \mathcal{N}(X^\top \mathbf{w}, \sigma_\varepsilon^2 I), \end{aligned} \tag{2.3}$$

where $|\mathbf{z}|$ is the Euclidean length of vector $\mathbf{z}$. In the Bayesian formalism, a prior is set over the parameters. We specify that

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p), \tag{2.4}$$

where $\Sigma_p$ is a covariance matrix. Using Bayes' rule, we have that the

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}},$$

meaning that

$$p(\mathbf{w}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{y})}. \tag{2.5}$$

Inference in the Bayesian linear model is based on the posterior distribution over the weights where the normalizing constant, or marginal likelihood, is independent of the weights. The *marginal likelihood* is given by

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{w})p(\mathbf{w})d\mathbf{w}, \tag{2.6}$$

and the *posterior* follows a Gaussian distribution such that

$$p(\mathbf{w}|\mathbf{y}) \sim \mathcal{N}\left(\frac{1}{\sigma_\varepsilon^2}A^{-1}X\mathbf{y}, A^{-1}\right), \tag{2.7}$$

where $A = \sigma_\varepsilon^{-2}XX^\top + \Sigma_p^{-1}$. The mean of the posterior distribution is also its mode and is known as the *maximum a posteriori* ($MAP$) estimate of $\mathbf{w}$.

Taking the average over all possible parameter values weighted by their posterior probabilities allows us to make predictions for a test case. The predictive distribution for $f_* \stackrel{\text{def}}{=} f(\mathbf{x}_*)$ at $\mathbf{x}_*$ is given by averaging the output of all possible linear models with respect to the Gaussian posterior:

$$\begin{aligned} p(f_*|\mathbf{x}_*, \mathbf{y}) &= \int p(f_*|\mathbf{x}_*, \mathbf{w})p(\mathbf{w}|\mathbf{y})d\mathbf{w} \\ &= \mathcal{N}\left(\frac{1}{\sigma_\varepsilon^2}\mathbf{x}_*^\top A^{-1}X\mathbf{y}, \mathbf{x}_*^\top A^{-1}\mathbf{x}_*\right). \end{aligned} \tag{2.8}$$

### 2.1.2 Projection into a Feature Space

To overcome the limited expressiveness of Bayesian linear models, we can apply linear models in higher dimensional spaces. Let $\boldsymbol{\phi}(\mathbf{x})$ be a function that maps a $D$-dimensional input vector $\mathbf{x}$ into an $M$-dimensional feature space. This projection must use a set of basis functions that is composed of fixed functions (functions that are independent of the parameters $\mathbf{w}$) to ensure that the model is linear in its parameters. Then, we have

$$f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}, \tag{2.9}$$

where the vector of parameters now has length $M$. Now, let $\Phi(X)$ be the $M \times N$ matrix of columns of $\boldsymbol{\phi}(\mathbf{x})$ for all cases in the training set. By denoting $\Phi = \Phi(X)$ and $A = \sigma_\varepsilon^{-2}\Phi\Phi^\top + \Sigma_p^{-1}$, the predictive distribution becomes

$$f_*|\mathbf{x}_*, \mathbf{y} \sim \mathcal{N}\left(\frac{1}{\sigma_\varepsilon^2}\boldsymbol{\phi}(\mathbf{x}_*)^\top A^{-1}\Phi\mathbf{y}, \boldsymbol{\phi}(\mathbf{x}_*)^\top A^{-1}\boldsymbol{\phi}(\mathbf{x}_*)\right). \tag{2.10}$$

To make predictions, we need to invert matrix $A$ of size $M \times M$. If we use the shorthand expression $\boldsymbol{\phi}(\mathbf{x}_*) = \boldsymbol{\phi}_*$ and define $K = \Phi^\top\Sigma_p\Phi$, then Equation (2.10) can be reformulated as

$$\begin{aligned} f_*|\mathbf{x}_*, \mathbf{y} \sim \mathcal{N}\Big(&\boldsymbol{\phi}_*^\top\Sigma_p\Phi(K + \sigma_\varepsilon^2 I)^{-1}\mathbf{y}, \\ &\boldsymbol{\phi}_*^\top\Sigma_p\boldsymbol{\phi}_* - \boldsymbol{\phi}_*^\top\Sigma_p\Phi(K + \sigma_\varepsilon^2 I)^{-1}\Phi^\top\Sigma_p\boldsymbol{\phi}_*\Big). \end{aligned} \tag{2.11}$$

When the feature space is present in Equation (2.11), it takes the form $\Phi^\top\Sigma_p\Phi$, $\boldsymbol{\phi}_*^\top\Sigma_p\Phi$, or $\boldsymbol{\phi}_*^\top\Sigma_p\boldsymbol{\phi}_*$. Thus, the entries of these matrices all take the form

$$\boldsymbol{\phi}(\mathbf{x})^\top\Sigma_p\boldsymbol{\phi}(\mathbf{x}'),$$

where $\mathbf{x}$ and $\mathbf{x}'$ are from either the training set or test set. Now we define

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top\Sigma_p\boldsymbol{\phi}(\mathbf{x}') \tag{2.12}$$

as a kernel or covariance function. If $\Sigma_p$ is positive definite, then we can represent the kernel as a dot product such that $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}(\mathbf{x}) \cdot \boldsymbol{\psi}(\mathbf{x}')$ with $\boldsymbol{\psi}(\mathbf{x}) = \Sigma_p^{1/2} \boldsymbol{\phi}(\mathbf{x})$, where $(\Sigma_p^{1/2})^2 = \Sigma_p$. In such case, we can use the *kernel trick* by computing the inner products in the feature space.

In the weight-space view, GPs can be defined as a parameterized function that follows a Gaussian distribution. An alternative but equivalent perspective for GPs is possible by considering inference directly in the function space. The equivalence of these two perspectives will be verified in the next section.

## 2.2 Function-space View

A GP describes a distribution over functions.

**Definition 2.2.1.** [49, Sec. 2.2] A **Gaussian process** is a collection of random variables, any finite number of which have a joint Gaussian distribution.

A GP is specified by its mean and covariance functions. The mean function $m(\mathbf{x})$ and covariance (kernel) function $k(\mathbf{x}, \mathbf{x}')$ are defined as

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{2.13}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \tag{2.14}$$

and GPs are written as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.15}$$

The covariance function $\text{cov}(f(\mathbf{x}_p), f(\mathbf{x}_q)) = k(\mathbf{x}_p, \mathbf{x}_q)$, for indices $p$ and $q$ and kernel function $k$, specifies the covariance between between pairs of random variables. Note that the covariance between the outputs is a function of the inputs and that the

specification of the covariance function implies a distribution over functions. This allows for the incorporation of knowledge that the training data provides about a function.

The definition of a GP implies the requirement of *consistency*, also known as the *marginalization property*. This means that the examination of a set of variables does not change the distribution of a subset of the variables. That is, a GP satisfies the property that if $(y_1, y_2)^\top \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, then $y_1 \sim \mathcal{N}(\mu_{11}, \Sigma_{11})$. This condition is automatically satisfied when the covariance function is specified by the covariance matrix [49].

We let $f_i \stackrel{\text{def}}{=} f(\mathbf{x}_i)$, where $\mathbf{x}_i$ is the input corresponding to the case $(\mathbf{x}_i, y_i)$. When there are $N$ training points and $N_*$ test points, then $K(X, X_*)$ denotes the $N \times N_*$ matrix of covariances for all pair of training and test points. $K(X_*, X)$ and $K(X, X)$ are defined analogously. Let $\mathbf{f}_*$ denote the GP posterior prediction for a test point $\mathbf{x}_*$. Then, since $X$ and $X_*$ are non-random, the noise-free predictive distribution is given by

$$\begin{aligned} \mathbf{f}_* | \mathbf{f} \sim \mathcal{N}(&K(X_*, X)K(X, X)^{-1}\mathbf{f}, \\ &K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)). \end{aligned} \tag{2.16}$$

In practice, we model situations with noise. By assuming $\varepsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\varepsilon^2)$, the prior on noisy observations becomes

$$\operatorname{cov}(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_\varepsilon^2 \delta_{pq} \quad \text{or} \quad K_{\mathbf{y}} \stackrel{\text{def}}{=} \operatorname{cov}(\mathbf{y}) = K(X, X) + \sigma_\varepsilon^2 I, \tag{2.17}$$

where $\delta_{pq}$ is the Kronecker delta that is one if and only if $p = q$ and zero otherwise. The identity matrix is included because of our assumption about the independence of noise. It follows that the joint distribution of the observed target values and the

function values under the prior is

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_\varepsilon^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2.18}$$

By Theorem A.0.1 in Appendix A, we finally arrive at the predictive equations for GP regression which are

$$\mathbf{f}_*|\mathbf{y} \sim \mathcal{N}(\bar{\mathbf{f}}_*, \operatorname{cov}(\mathbf{f}_*)), \quad \text{where} \tag{2.19}$$

$$\bar{\mathbf{f}}_* \stackrel{\text{def}}{=} \mathbb{E}[\mathbf{f}_*|\mathbf{y}] = K(X_*,X)[K(X,X) + \sigma_\varepsilon^2 I]^{-1}\mathbf{y}, \tag{2.20}$$

$$\operatorname{cov}(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)[K(X,X) + \sigma_\varepsilon^2 I]^{-1}K(X,X_*). \tag{2.21}$$

This gives us a correspondence to the weight-space view by relating it to Equation (2.11) and identifying that $K(C,D) = \Phi(C)^\top \Sigma_p \Phi(D)$, where $C$ and $D$ are either $X$ or $X_*$.

We can compactify our notation for a single test point $\mathbf{x}_*$ if we let $K = K(X,X)$ and $\mathbf{k}_* = \mathbf{k}(\mathbf{x}_*)$. Then the conditional distribution for a test point $\mathbf{x}_*$ is

$$\overline{f}_* = \mathbf{k}_*^\top (K + \sigma_\varepsilon^2 I)^{-1}\mathbf{y} \tag{2.22}$$

$$\operatorname{var}(f_*) = k(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_\varepsilon^2 I)^{-1}\mathbf{k}_* \tag{2.23}$$

Note that the variance depends only on the inputs and not the observed targets. The predictive distribution of $\mathbf{y}_*$ can be obtained by adding $\sigma_\varepsilon^2 I$ to $\operatorname{cov}(\mathbf{f}_*)$ given by Equation (2.21).

### 2.2.1 Marginal Likelihoods

The *marginal likelihood* (or *evidence*) $p(\mathbf{y})$ is the integral of the likelihood times the prior:

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})d\mathbf{f}. \tag{2.24}$$

The term *marginal* refers to the marginalization over $\mathbf{f}$. Under the GP, the prior is Gaussian with $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K)$. It follows that

$$\log p(\mathbf{f}) = -\frac{1}{2}\mathbf{f}^\top K^{-1}\mathbf{f} - \frac{1}{2}\log|K| - \frac{N}{2}\log 2\pi. \tag{2.25}$$

The likelihood follows a factorized Gaussian distribution such that $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_\varepsilon^2 I)$. The log marginal likelihood is obtained by observing that $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, K_\mathbf{y})$, giving us

$$\log p(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^\top K_\mathbf{y}^{-1}\mathbf{y} - \frac{1}{2}\log|K_\mathbf{y}| - \frac{N}{2}\log 2\pi. \tag{2.26}$$

## 2.3 GPs with GPyTorch

To perform our analysis in this project, we will be using GPyTorch [24] to build and train GPs. This is a free resource that operates on the PyTorch tensor framework [46] and has many options for customization.

### 2.3.1 GP Initialization and Training

Initializing a GP in GPyTorch is made relatively simple because of all the examples available in the documentation. Prior to construction, the user needs to specify the mean and covariance functions. Then the user creates likelihood and model objects in order to instantiate the GP.

Once the GP is initialized, the GP is trained to find the optimal model hyper-parameters, where the number of training iterations is specified by the user. The default is for GPyTorch to use the marginal log likelihood as the loss function in conjunction with the Adam optimizer [34] during model training to optimize model performance, and these are what we use in this thesis. After the training iterations are complete, the GP is set into evaluation mode so that the distribution object

contains the posterior mean and covariance. The GP can then fit on the test data and provide predictive distributions on novel inputs.

# Chapter 3

# Kernels

Kernel methods are an approach to pattern analysis that maps data into a suitable feature space and then uses algorithms to discover linear patterns in the embedded data [57]. Kernels are significant in our context because the choice of kernel determines how a GP learns from training data. Much of the theoretical information contained in this chapter is referenced from this Rasmussen and Williams [49] and Shawe-Taylor et al. [57].

## 3.1 Kernel Functions

Let $\boldsymbol{\phi}$ be a mapping from $\mathbb{R}^D$ to an inner product feature space $F$ such that

$$\boldsymbol{\phi} : \mathbf{x} \longmapsto \boldsymbol{\phi}(\mathbf{x}) \in F \tag{3.1}$$

for $\mathbf{x} \in \mathbb{R}^D$.

**Definition 3.1.1.** [57, sec. 2.2] A **kernel function** is a function $k$ that for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$ satisfies

$$k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle, \tag{3.2}$$

where $\langle \cdot, \cdot \rangle$ is an inner product.

The initial mapping component is performed implicitly. Moreover, the choice of kernel function depends on the type of data being examined, the task at hand, and prior knowledge about the data.

Let $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\{\mathbf{z}_1, \ldots, \mathbf{z}_{\tilde{N}}\}$ be sets of vectors. Now let $k$ be a kernel function used to evaluate inner products in a feature space with feature map $\boldsymbol{\phi}$.

**Definition 3.1.2.** [57, sec. 3.1] A **Gram matrix** is defined as the square matrix $K$ with entries

$$K_{ij} = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle = k(\mathbf{x}_i, \mathbf{x}_j). \tag{3.3}$$

Gram matrices are composed of pairwise inner products. Since inner products are symmetric and positive definite, it follows that Gram matrices are symmetric positive semi-definite (PSD) [57, sec. 3.1]. In general, we define a *kernel matrix* as the $N \times \tilde{N}$ matrix whose entries are

$$K_{ij} = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{z}_j) \rangle = k(\mathbf{x}_i, \mathbf{z}_j). \tag{3.4}$$

In machine learning, it is generally assumed that the kernel functions being used are covariance functions. If $k$ is a covariance function and $K$ is a Gram matrix, then we call $K$ a *covariance matrix* [49, sec. 4.2].

## 3.2 Common Kernels

In this section, we introduce positive-definite kernels which qualify as covariance functions that will be used in the investigative portion of this thesis. Once again, let $\mathbf{x}$ and $\mathbf{x}'$ be $D$-dimensional vectors.

## Constant kernel

The *constant kernel* is simple. It is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0^2, \tag{3.5}$$

for $\sigma_0 \geq 0$ [49, sec. 4.2]. As will be explained in Section 3.3, this kernel is important because it will serve as a building block for other kernels.

## Polynomial Kernel

The *polynomial kerne*l, as described by Shawe-Taylor et al. [57], is defined for a kernel $\tilde{k}$ as

$$k(\mathbf{x}, \mathbf{x}') = p(\tilde{k}(\mathbf{x}, \mathbf{x}')), \tag{3.6}$$

where $p(\cdot)$ is any polynomial with positive coefficients. It is often used in the special case

$$k_d(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + \sigma_0^2)^d, \tag{3.7}$$

where $d$ is the degree hyperparameter and $\sigma_0^2$ is an offset hyperparameter. The specific case

$$k_1(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle + \sigma_0^2 \tag{3.8}$$

is known as the *linear kernel*.

## Matérn Kernel

The class of *Matérn kernels* as described by Rasmussen and Williams [49] is given by

$$k_{\text{Matérn}}(\mathbf{r}) = \frac{2^{1-\nu}}{\Gamma(\nu)} \Big( \frac{\sqrt{2\nu}\mathbf{r}}{\ell} \Big)^{\nu} K_{\nu} \Big( \frac{\sqrt{2\nu}\mathbf{r}}{\ell} \Big), \tag{3.9}$$

where $\nu$ is a positive smoothness hyperparameter, $\ell$ is a positive lengthscale hyperparameter, and $K_\nu$ is a modified Bessel function. In practice, when applying this covariance function, we argue $\mathbf{r} = |\mathbf{x} - \mathbf{x}'|$. Since it is a function of $\mathbf{x} - \mathbf{x}'$, the kernel is *stationary* such that it is invariant to translations in the input space. Furthermore, since it is a function of $|\mathbf{x} - \mathbf{x}'|$, the kernel is *isotropic* such that it is invariant to all rigid motions [49].

The Matérn covariance function becomes simple when $\nu$ is a half integer such that $\nu = p + 1/2$ for some non-negative integer $p$. In this case, the function is a product of an exponential and a polynomial of order $p$, giving

$$k_{\nu=p+1/2}(\mathbf{r}) = \exp\Big(-\frac{\sqrt{2\nu}\mathbf{r}}{\ell}\Big)\frac{\Gamma(p+1)}{\Gamma(2p+1)}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}\Big(\frac{\sqrt{8\nu}\mathbf{r}}{\ell}\Big)^{p-i}. \tag{3.10}$$

The special case when $p = 0$ and $\nu = 1/2$ gives the *exponential covariance function*

$$k_{\nu=1/2}(\mathbf{r}) = \exp\Big(-\frac{\mathbf{r}}{\ell}\Big). \tag{3.11}$$

Interesting cases also arise when $p = 1$ and $p = 2$, giving us

$$k_{\nu=3/2}(\mathbf{r}) = \Big(1 + \frac{\sqrt{3}\mathbf{r}}{\ell}\Big)\exp\Big(-\frac{\sqrt{3}\mathbf{r}}{\ell}\Big) \tag{3.12}$$

and

$$k_{\nu=5/2}(\mathbf{r}) = \Big(1 + \frac{\sqrt{5}\mathbf{r}}{\ell} + \frac{5\mathbf{r}^2}{3\ell^2}\Big)\exp\Big(-\frac{\sqrt{5}\mathbf{r}}{\ell}\Big). \tag{3.13}$$

Choosing a higher value of $\nu$ results in a smoother process, while choosing a lower value of $\nu$ results in a more jagged, rough process [49]. When selecting a value of $\nu$, it is not as effective to choose values much greater than $\nu = 7/2$ for it becomes difficult to distinguish between values from finite noisy training examples [49].

### Radial Basis Function Kernel

The *radial basis function* (*RBF*) *kernel*, also known as the *squared exponential kernel* or *Gaussian kernel*, is a special case of the Matérn kernel in which we impose

that $\nu \to \infty$. The radial basis function kernel is defined as

$$k_{\text{RBF}}(\mathbf{r}) = \exp\Big(-\frac{\mathbf{r}^2}{2\ell^2}\Big). \tag{3.14}$$

Despite the strong smoothness assumption, this is likely the most common kernel used in the kernel machine field [49].

For kernels like the RBF kernel that have a lengthscale hyperparameter, there is the option to implement a separate lengthscale for each dimension of the training input. That is, if the design matrix is $D \times N$, then the GP can optionally implement $D$ *characteristic lengthscales*. This is known as *automatic relevance determination* (*ARD*) and can potentially be used to identify dimensions that contribute more information to the model or to effectively remove weak contributors from inference. If ARD is applied to the RBF kernel, for example, then the kernel becomes a product and takes the form

$$k_{\text{RBF}}(\mathbf{r}) = \prod_{i=1}^{D} \exp\Big(-\frac{r_i^2}{2{\ell_i}^2}\Big), \tag{3.15}$$

where $r_i = |x_i - x_i'|$ and there are $D$ characteristic lengthscales $\ell_i$.

## 3.3 Compounding Kernels

Kernels satisfy closure properties that allow for the creation of more complicated kernels. The following theorem from Shawe-Taylor et al. [57] illustrates this relationship.

**Theorem 3.3.1.** [57, sec. 3.4] Let $k_1$ and $k_2$ be kernels over $\mathbb{R}^D \times \mathbb{R}^D$, $c \in \mathbb{R}^+$, and $B$ a symmetric positive semi-definite $D \times D$ matrix. Then the following are valid

kernels:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}'), \tag{3.16}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}'), \tag{3.17}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}'), \tag{3.18}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}'B\mathbf{x}. \tag{3.19}$$

In simple terms, the rescaling of kernels by a positive value, addition of kernels, product of kernels, and the quadratic form with the inclusion of a symmetric PSD matrix are all valid kernels. In this thesis, we will refer to the actions of adding and multiplying kernels as the *compounding* of kernels since it combines the properties of the component kernels.

Duvenaud [16] explored the behavior of GP models that are equipped with compound kernels. Some observations pertaining to the multiplication of kernels were that multiplying $p$ linear kernels produces a polynomial kernel of degree $p$, multiplying by a RBF kernel changes global structure to local structure, and multiplying by a linear kernel produces the behavior of growing amplitude for the marginal standard deviation of the function. On the other hand, some observations pertaining to the addition of kernels were that adding a periodic kernel contributed periodic behavior to the function, adding the RBF kernel contributed variation to the function, and adding RBF kernels with different lengthscales introduced "slow and fast variation." The work done by Duvenaud [16] suggests that unique kernels can be compounded from base kernels that exhibit different behavior and are able to detect different patterns within data.

## 3.4 Hyperparameter Optimization

One method to optimize hyperparameters is with grid search, a process in which a Cartesian product of subsets of choices for hyperparameters are used for distinct models, and their performances are compared. This algorithm requires selecting a performance metric, and then evaluating the model performance on a validation set or with cross-validation (see Chapter 6) to determine which specification of hyperparameter values results in a superior model. After doing this, the grid can be refined to a finer grid around the set of values that produced the best model. Reiterating this process has the potential to lead to better models [29].

An alternative to grid search is the random search algorithm. This process involves setting a distribution among the space of possible hyperparameters and then randomly sampling from this until a favorable model is obtained. This method has the capacity to outperform grid search and has the benefits that it can be parallelized and does not need to commit to a set of experiments [8].

The grid and random search algorithms can become inefficient when the number of hyperparameters or model variations grows large. Beck [6] describes how GPs can efficiently optimize hyperparameters by maximizing the marginal likelihood given by Equation (2.24) with respect to the full training data in what is known as an "empirical Bayes" or "evidence procedure." The method entails defining gradients of the marginal likelihood and then employing gradient ascent or descent optimizers to locate extrema. In this procedure, the marginal likelihood incorporates the full set of hyperparameters $\boldsymbol{\theta}$, but the resulting marginal log likelihood $\log p(\mathbf{y}|\boldsymbol{\theta})$ takes the same form as Equation (2.26).

Assuming $K_{\mathbf{y}} = \text{cov}(\mathbf{y})$ as given by Equation (2.17), the optimization step requires taking the likelihood derivatives with respect to the hyperparameters $\boldsymbol{\theta}$,

which gives

$$\frac{\partial \log p(\mathbf{y}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{2} \operatorname{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - K_{\mathbf{y}}^{-1})\frac{\partial K_{\mathbf{y}}}{\partial \boldsymbol{\theta}}\right), \tag{3.20}$$

where $\boldsymbol{\alpha} = K_{\mathbf{y}}^{-1}\mathbf{y}$ and the derivatives of $K_{\mathbf{y}}$ depend on the kernel being used. Thus, the kernel can be optimized as long as its gradient vector is defined.

It should be noted that the marginal likelihood is not necessarily convex, so it is possible to encounter local extrema when doing this process. This potential problem can be approached by building and training multiple models with different hyperparameter initializations [6].

## 3.5 Kernels with GPyTorch

GPyTorch [24] allows the user to equip a GP with a kernel of their choice. Some kernels require the user to specify hyperparameters such as the power hyperparameter for the polynomial kernel, and others have default hyperparameters that the user can set such as the smoothness hyperparameter for the Matérn kernel. Additionally, some kernels have option to use multiple characteristic lengthscale hyperparameters with ARD. GPyTorch also permits the user to specify priors or constraints on kernel hyperparameters to help with optimization.

A kernel in GPyTorch is optimized as the GP equipped with a kernel is trained. The GP computes the loss using the loss function to adjust the kernel hyperparameter after each training iteration. One feature in GPyTorch is that the user may inspect the hyperparameters of the model. This can be useful to see how hyperparameters change with each training iteration or after training is complete.

Among the many benefits of using GPyTorch, the user is able is to create custom kernels. GPyTorch makes it relatively easy to implement and register hyperparam-

eters. This will become important for us later when we introduce the string kernel.

# Chapter 4

# Multi-task GPs and Linear Models of Coregionalization

Multi-task learning is a type of modeling that employs joint prediction while exploiting the interactions and information shared between tasks. [4]. Multi-task GPs have the potential to learn more from multiple tasks than single-task GPs operating independently.

## 4.1 Separable Kernels

Multi-output kernel functions can be expressed through *separable kernels* that decouple the contribution of the input and output [4]. Let $T$ be the number of tasks that are being modeled. Now let $k$ and $k_T$ be scalar kernels on $X \times X$ and $\{1, \ldots, T\} \times \{1, \ldots, T\}$, respectively. A separable kernel takes the form

$$k((\mathbf{x}, t), (\mathbf{x}', t')) = k(\mathbf{x}, \mathbf{x}')k_T(t, t'), \tag{4.1}$$

which is a valid kernel by Theorem 3.3.1. In general, the matrix representation for this kernel is

$$(K(\mathbf{x}, \mathbf{x}'))_{t,t'} = k(\mathbf{x}, \mathbf{x}')k_T(t, t'), \tag{4.2}$$

which can equivalently be expressed as

$$K(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}')B, \tag{4.3}$$

where $B$ is a $T \times T$ symmetric PSD matrix [4].

More generally, we have the class of kernels given by

$$K(\mathbf{x}, \mathbf{x}') = \sum_{q=1}^{Q} k_q(\mathbf{x}, \mathbf{x}')B_q, \tag{4.4}$$

which is fittingly called *sum of separable kernels*. For this class of kernels, the kernel matrix associated to a dataset $X$ can be written as

$$K(X, X) = \sum_{q=1}^{Q} B_q \otimes k_q(X, X). \tag{4.5}$$

where $\otimes$ is the Kronecker product. The off-diagonal elements of $B$ encodes the dependencies among the outputs. As such, in the simplest case where $B = I$, the outputs are all treated as being independent, and the associated kernel matrix $K(X, X)$ is block diagonal.

## 4.2 Multi-task GPs

For the multi-task case, first let $Y$ be the $N \times T$ matrix such that each of the $N$ rows in the matrix corresponds to the vector of responses for the $T$ tasks. We index the output matrix so that $y_{it}$ corresponds to the response for the $t$th task on the $i$th input $\mathbf{x}_i$. Now, let us arrange the columns of $Y$ as a vector such that

$$\mathbf{y} = (y_{11}, \ldots, y_{N1}, y_{12}, \ldots, y_{N2}, \ldots, y_{1T}, \ldots, y_{NT})^\top. \tag{4.6}$$

We now define a set of latent GPs $\{f_i : i = 1, \ldots, T\}$, one for each task. The multi-task GP model proposed by Bonilla et al. [9] tries to "learn inter-task dependencies based solely on the *task identities* and the observed data for each task." The GP model learns a "free-form" task-similarity matrix that is used with a parameterized covariance function over the input features. The multi-task model places a GP prior over the latent functions $\{f_1, \ldots, f_T\}$ so that correlations can be directly induced between tasks. For this model, we assume that the GP has a zero mean. Now we set

$$\text{cov}[f_t(\mathbf{x}), f_{t'}(\mathbf{x}')] = K^f_{tt'} k^x(\mathbf{x}, \mathbf{x}'), \tag{4.7}$$

where $K^f$ is a PSD matrix that specifies the inter-task similarities between each pair $(f_t, f_{t'})$ and $k^x$ is a covariance function over the inputs. Then we continue to set

$$y_{it} \sim \mathcal{N}(f_t(\mathbf{x}_i), \sigma_t^2), \tag{4.8}$$

where $\sigma_t^2$ is the noise variance for the $t$th task. It is important that the joint Gaussian distribution over $\mathbf{y}$ is not block-diagonal with respect to the tasks so that the tasks influence one another at time of prediction [9].

The model presented here by Bonilla et al. [9] is a case of the semiparametric latent factor model (SLFM) provided by Teh et al. [62]. The SLFM model uses $P$ latent processes, where $P \leq T$, and each process has its own covariance function. Obtaining the noiseless outputs requires a linear mixing of the processes with a $T \times P$ matrix $\Phi$. Since the covariance matrix of the system has at most rank $NP$, choosing $P < T$ results in a degenerate GP. The model we use is a simple case of the SLFM model because each process shares the same covariance function and sets $K^f = \Phi\Phi^\top$, resulting in less hyperparameters and reducing the chance of overfitting. Note that if one were to choose $P > T$, it is still possible to find an

$T \times T$ matrix $\Phi'$ such $\Phi'\Phi'^{\top} = \Phi\Phi^{\top}$ [62].

### 4.2.1 Inference

The mean prediction on a new data-point $\mathbf{x}_*$ for a task $t$ is given by [9]

$$\bar{f}_t(\mathbf{x}_*) = (\mathbf{k}_t^f \otimes \mathbf{k}_*^x)^{\top}\Sigma^{-1}\mathbf{y}, \tag{4.9}$$

where $\otimes$ denotes the Kronecker product, $\mathbf{k}_t^f$ selects the $t$th column of $K^f$, $\mathbf{k}_*^x$ is the vector of covariances between the test point $\mathbf{x}_*$ and the training points, $K^x$ is the matrix of covariances between all pairs of training points, $D$ is a $T \times T$ diagonal matrix with $D_{tt} = \sigma_t^2$, and

$$\Sigma = K^f \otimes K^x + D \otimes I \tag{4.10}$$

is a $NT \times NT$ covariance matrix.

It is noted that if there is no noise in the data such that $D = \mathbf{0}$, then the tasks are conditionally independent and there is no transfer between tasks. This happens because the inter-task similarity matrix $K^f$ is the sample covariance of the de-correlated $Y$ matrix. In this case, the inter-task relationship is still technically maintained because of the sharing of the kernel function across tasks [9].

## 4.3 Linear Models of Coregionalization

Consider a set of $T$ outputs $\{f_t(\mathbf{x})\}_{t=1}^T$ with $\mathbf{x} \in \mathbb{R}^D$. In the *linear model of coregionalization* (*LMC*), as formulated by Alvarez et al. [4], each component is expressed as

$$f_t(\mathbf{x}) = \sum_{q=1}^{Q} a_{t,q} u_q(\mathbf{x}), \tag{4.11}$$

where the latent functions $u_q(\mathbf{x})$ have mean zero and covariance $\text{cov}[u_q(\mathbf{x}), u_{q'}(\mathbf{x}')] = k_q(\mathbf{x}, \mathbf{x}')\delta_{qq'}$ and $a_{t,q}$ are scalar coefficients. In this formulation, the processes $\{u_q(\mathbf{x})\}_{q=1}^{Q}$ are independent for $q \neq q'$. The expression $f_t(\mathbf{x})$ can be rewritten by grouping the functions $u_q(\mathbf{x})$ into $Q$ groups, where the functions within each group shares the same covariance function but are independent. This becomes

$$f_t(\mathbf{x}) = \sum_{q=1}^{Q} \sum_{i=1}^{R_q} a_{t,q}^i u_q^i(\mathbf{x}), \tag{4.12}$$

where the functions $u_q^i(\mathbf{x})$ have mean zero and covariance $\text{cov}[u_q^i(\mathbf{x}), u_{q'}^{i'}(\mathbf{x}')] = k_q(\mathbf{x}, \mathbf{x}')\delta_{qq'}\delta_{ii'}$. From this, we see that the outputs are expressed as linear combinations of independent GPs. By independence of the functions $u_q^i(\mathbf{x})$, the covariance between two functions $f_t(\mathbf{x})$ and $f_{t'}(\mathbf{x})$ is given by

$$\begin{aligned} \text{cov}[f_t(\mathbf{x}), f_{t'}(\mathbf{x}')] &= (K(\mathbf{x}, \mathbf{x}'))_{t,t'} \\ &= \sum_{q=1}^{Q} \sum_{i=1}^{R_q} a_{t,q}^i a_{t',q}^i k_q(\mathbf{x}, \mathbf{x}') \\ &= \sum_{q=1}^{Q} b_{t,t'}^q k_q(\mathbf{x}, \mathbf{x}'), \end{aligned} \tag{4.13}$$

where $b_{t,t'}^q = \sum_{i=1}^{R_q} a_{t,q}^i a_{t',q}^i$. The kernel $K(\mathbf{x}, \mathbf{x}')$ can then be expressed as

$$K(\mathbf{x}, \mathbf{x}') = \sum_{q=1}^{Q} B_q k_q(\mathbf{x}, \mathbf{x}'), \tag{4.14}$$

where each $B_q \in \mathbb{R}^{T \times T}$ is known as a *coregionalization matrix* [4]. The elements of each $B_q$ are the coefficients $b_{t,t'}^q$ in Equation (4.13), and the rank for each $B_q$ is determined by the number of latent functions that share the same covariance function $k_q(\mathbf{x}, \mathbf{x}')$. Thus, if each of the tasks are pairwise independent, then $B = I_T$.

The LMC constitutes a sum of separable kernels that represents the covariance function as a sum of products of two covariance functions: one that models the

dependence between the outputs ($B_q$) and one that models the input dependence ($k_q(\mathbf{x}, \mathbf{x}')$). The former is independent of the input vector $\mathbf{x}$, while the latter is independent of the functions $\{f_t(\mathbf{x})\}$ [4].

### 4.3.1 Inference

Let $\boldsymbol{\theta}$ is the set of hyperparameters for the covariance function $K(\mathbf{x}, \mathbf{x}')$. Now let $K(X, X)$ be the $NT \times NT$ covariance matrix given by Equation (4.5) with entries $(K(\mathbf{x}_i, \mathbf{x}_j))_{t,t'}$ for $i, j = 1, \ldots, N$ and $t, t' = 1, \ldots, T$. Then, the predictive distribution for a test input is $\mathbf{x}_*$ [4]

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{f}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{f}_*(\mathbf{x}_*), K_*(\mathbf{x}_*, \mathbf{x}_*)), \tag{4.15}$$

with

$$\mathbf{f}_*(\mathbf{x}_*) = K_{\mathbf{x}_*}^\top (K(X, X) + \Sigma)^{-1} \overline{\mathbf{y}}, \tag{4.16}$$

$$K_*(\mathbf{x}_*, \mathbf{x}_*) = K(\mathbf{x}_*, \mathbf{x}_*) - K_{\mathbf{x}_*}(K(X, X) + \Sigma)^{-1} K_{\mathbf{x}_*}^\top, \tag{4.17}$$

where $K_{\mathbf{x}_*} \in \mathbb{R}^{T \times NT}$ has entries $(K(\mathbf{x}_*, \mathbf{x}_j))_{t,t'}$ for $j = 1, \ldots, N$ and $t, t' = 1, \ldots, T$, $\Sigma = D \otimes I_N$ such that $D$ is the diagonal matrix of noise variances with $D_{tt} = \sigma_t^2$, and $\overline{\mathbf{y}}$ is an arrangement of the rows of $Y$ as a vector such that

$$\overline{\mathbf{y}} = (y_{11}, \ldots, y_{1T}, y_{21}, \ldots, y_{2T}, \ldots, y_{N1}, \ldots, y_{NT})^\top. \tag{4.18}$$

## 4.4 Multi-task GPs and LMC with GPyTorch

Both multi-task GPs and LMC are available to implement in GPyTorch [24]. These make for powerful implementations of GPs that are versatile and highly customizable.

Multi-task kernels in GPyTorch are relatively easy to implement. Upon initialization of a multi-task GP, in addition to the parameters needed to initialize a single-task GP (see Section 2.3), GPyTorch requires that the user specifies the number of tasks $T$ and the rank of kernel being equipped, where the rank must not exceed $T$. There is an option for the user to not only inspect hyperparameters, but also to get the inter-task covariance matrix associated with GP. This is useful if the user is trying to inspect the inter-task relationships.

There is support for the LMC kernel as well. Using this kernel requires that the user specifies a list of kernels with their respective ranks (ranks must be in $[0, T]$) and the rank of the noise matrix being used in the process. If the noise is set to 0, then GPyTorch will assume that the matrix is diagonal and that all the noises are independent.

### 4.4.1 ModeList Multi-Output GPs

The *ModelList multi-output GP* regression model is a special application of GP offered by GPyTorch [24]. This model method allows the user to implement multiple independent GPs in a single multi-output GP using a *ModelList*. Unlike the multi-task case, ModelList does not model correlations between outcomes and treats each GP separately. This model has the advantage that the GPs train across all tasks simultaneously and can train a set of independent single-task GPs relatively quickly. GPyTorch recommends using this kind of model when the number of training and test points is different for each of the outcomes or if it uses different covariance functions or likelihoods for each outcome. The ModelList optimizes the kernels' hyperparameters by using the sum of the marginal log-likelihoods for each task.

# Chapter 5

# Strings in NLP

Firth [20] wrote, "you shall know a word by the company it keeps." This speaks about the distributional structure of language which posits that related words occur in similar contexts [27]. In NLP, a machine learning model uses *embeddings*, representations of meanings of words learned directly from their distributions in text [33].

Using kernel methods for NLP is the operation of defining kernel functions to model linguistic phenomena [23] and is a matter of choosing the right embedding. In this chapter, we will provide a rigorous formulation of string sequences, discuss embedding approaches that are used with kernels, and introduce the string kernel.

## 5.1 Definitions and Notation

**Definition 5.1.1.** [39] Let $\Sigma$ be a finite set of characters which we will call an *alphabet*. A **string** is a finite sequence of characters from $\Sigma$, including the empty sequence.

Let $s$ and $s'$ be strings composed of characters from $\Sigma$. We now introduce the

following definitions and notation regarding strings which was introduced by Lodhi et al. [39]:

- The *length* of $s$ is denoted as $|s|$.
- The characters that compose $s$ will be subscripted by an index starting at 1 such that $s = s_1 s_2 \dots s_{|s|}$.
- The string obtained by concatenating $s$ and $s'$ is denoted $ss'$.
- For indices $i$ and $j$ satisfying $1 \leq i \leq j \leq |s|$, the string $s[i:j]$ is the *substring* of $s$ such that $s[i:j] = s_i \dots s_j$.
- The string $u$ is a *subsequence* of $s$ if there exists a tuple of indices $\mathbf{i} = (i_1, \dots, i_{|u|})$ satisfying $1 \leq i_1 < \dots < i_{|u|} \leq |s|$ such that $u_j = s_{i_j}$ for $j = 1, \dots, |u|$. The shorthand notation for this is $u = s[\mathbf{i}]$.
- The length of a subsequence $u$ of $s$ is $l(\mathbf{i}) = i_{|u|} - i_1 + 1$.
- An *n-gram* is a substring of length $n$ with characters from $\Sigma$ [40].
- The set of all strings of length $n$ is denoted as $\Sigma^n$.
- The set of all strings is denoted as

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n. \tag{5.1}$$

It should be mentioned that alphabets can be designated as any set of characters, where a character can be a single symbol or, more generally, any finite sequence of symbols. The definitions and notation above are relatively intuitive, except for the notion of a subsequence, which is deserving of examples. For the sake of readability, we will use double quotation marks to distinguish a string of characters. Starting

with a simple case, let $s =$ "formalism". The characters of $s$ can be indexed such that $s_1 =$ "f", $s_2 =$ "o", $\ldots, s_{|s|} = s_9 =$ "m". The substring $u =$ "form" is a subsequence of $s$ since $\mathbf{i} = (1, 2, 3, 4)$ is a vector of increasing indices such that $u = s[\mathbf{i}] = s[1, 2, 3, 4]$ and has length $l(\mathbf{i}) = 4 - 1 + 1 = 4$. Now, let us do a less simple example. Let $s =$ "circumlocution". Once again, the characters can be indexed such that $s_1 =$ "c", $s_2 =$ "i", $\ldots, s_{|s|} = s_{14} =$ "n". Then, $u =$ "motion" is a subsequence of $s$ since $\mathbf{i} = (6, 8, 11, 12, 13, 14)$ is a vector of increasing indices such that $u = s[\mathbf{i}] = s[6, 8, 11, 12, 13, 14]$ and has length $l(\mathbf{i}) = 14 - 6 + 1 = 9$. The importance of these computations will be made apparent when we introduce the string subsequence kernel.

## 5.2 Hard Matching

*Hard matching* is an approach to embeddings that utilizes "hard matching pattern rules" to extract information from text. The nature of these methods is to operate on the equivalence of strings and thus does not tolerate variation in lexical and syntactical constructions [68].

### 5.2.1 Bag of Words

The first reference to a bag of words model in linguistics was made by Harris [27], noting that "language is a tool" that has dynamic sets of rules. *Bag of words* is a model that counts the instances of *tokens* appearing in a corpus, capturing their frequencies. A token, in general, can be any string but is usually specified as words or word parts. The bag of words model is a relatively basic approach to NLP and is easy to implement [45]. In this model, word order is not maintained, which

means that a significant amount of semantic information can be lost from phrases or sentences. This model is worth introducing because it is referenced often in NLP literature and commonly used as a baseline of comparison when new methods are implemented [39, 42].

### 5.2.2 Character-level One-hot Encoding

Another naive approach to capture linguistic information is the *character-level one-hot encoding* scheme. One-hot encoding is a simple approach to encoding categorical data in which each categorical variable is assigned a bit for each state [26]. For our purposes, we use a modified version of this scheme that encode strings with a combination of unit vectors and zero vectors. The following procedure is a custom adaption from the common one-hot encoding scheme implemented by SciKit-Learn [56]:

Let the alphabet $\Sigma$ be a set of characters with a cardinality of at least one such that it must contain a positive number of distinct non-space symbols. Let $V$ be the cardinality of $\Sigma$. In this scheme, the space symbol is technically a valid character to include in a string but is reserved for signaling the separation of contiguous character sequences (words) and is not considered to be a character in the alphabet. It should be noted that making the space character not count as a character in the alphabet is intentional. This was done to maintain the notion that the space serves to separate words or morphemes.

In this scheme, *any* string composed of characters from $\Sigma$ is considered a word, even if it is nonsensical or contains non-letter characters. In general, it does not provide extra meaning to have contiguous space characters, so these can practically be replaced by a single space character. Moreover, space characters at the end

or beginning of a string do not add any information and can be removed without changing the meaning. In proceeding under these assumptions, it follows that for a string with $n$ single, non-end spaces that there are $n-1$ words in the string.

This scheme requires that we impose an ordering of the characters in $\Sigma$. Once this has been done, then we can define a one-to-one correspondence from each character to a unit vector in $\mathbb{R}^V$. As such, to represent a length-$n$ word, we can encode this in a $\mathbb{R}^{n\times V}$ matrix by stacking the transposed unit vectors. We will call this representation an *embedding matrix*. By design, we will let the zero vector correspond to the space character since it is not a character in $\Sigma$. Thus, to represent any string sequence of that may include space characters, we need only extend our encoding scheme accordingly to include the transposed zero vectors.

### Integer Encoding

The *integer encoding* is related to one-hot encoding but is a more efficient way to store the information. Instead of creating a function from $\Sigma$ to $\mathbb{R}^V$, we instead create a one-to-one correspondence from $\Sigma$ to $\{1,\ldots,V\}$. In this scheme, we maintain the ordering such that the first symbol in $\Sigma$ maps to 1, the second symbol in $\Sigma$ maps to 2, and so on, until the last symbol in $\Sigma$ maps to $V$. Finally, any space characters in a string map to 0. With this modification, any $n$-gram can be represented as a vector in $\{0,\ldots,V\}^n$. Furthermore, it is important to recognize that this vector has a one-to-one correspondence with a character-level one-hot encoding embedding matrix which can be done by mapping each element to the corresponding vectors in $\mathbb{R}^V$.

### 5.2.3 Word-Level Integer Encoding

The *word-level integer encoding* is another form of hard matching that will have the same scheme as the integer encoding described previously. The notion of a word is in this context is a set of characters that are separated by space characters, but now, words essentially are treated as characters in an alphabet in the same manner as the character-level one-hot encoding. Once again, the space character is special in that it is not considered a character in the alphabet.

The Keras package [61] made available by TensorFlow [2] can be utilized to make this encoding. The function uses a one-hot encoder and requires the user to specify a vocabulary size $V$. Once inputted, the words in each string are then mapped to an integer in $(0, V)$. Once again, it is the case that the 0-encoding has a special purpose and is a reserved value. One option in using the Keras one-hot encoder is to choose MD5 [52] as the hashing function. The main benefit of using this algorithm is that it is a "stable" hash function such that it reproduces the same results each time.

In a case where the vocabulary size is less than the number of unique words, clearly each word cannot be assigned a unique integer encoding. Nevertheless, the MD5 hash function is by no means perfect and is not guaranteed (in fact not likely) to assign a unique integer encoding to each word if the vocabulary size plus one matches the number of unique words. As a remark about word-level integer encodings, it is easy to see how the vocabulary size can grow large very fast. This suggests that it may suffer in practical value until sufficient data is provided that makes word re-occurrences prominent and patterns can emerge.

## 5.3 Soft Matching

*Soft matching* embedding approaches utilize soft pattern rules and probabilistic matching to extract information from text and allow for better accommodation of variability of expressions [68]. Distributional approaches to meaning acquisition utilize the correlation between distributional similarity to estimate meaning similarity [54].

### 5.3.1 GloVe Embeddings

*Global Vectors for Word Representation* (*GloVe*) are embeddings that are created with an unsupervised learning algorithm to obtain semantic representations of words using real-valued vectors [47]. GloVe embeddings are a free state-of-the-art resource that are used by many studying NLP [6, 51, 64, 65]. A user can freely download the code and a corpus from the website and proceed to obtain pre-trained word vectors by running the included script. The embeddings that we will use in this thesis comes from the corpus *Wikipedia 2014 + Gigaword 5*. The set of embeddings represents words that are all uncased (all lower case), contains 6B tokens, has a vocabulary size of 400K, and comes in 50d, 100d, 200d, and 300d vectors.

Pennington et al. [47] states that the embeddings are train on aggregated global word-word co-occurrence statistics and develop linear substructures in the word vector space. It is explained that GloVe is a log-bilinear regression model with a weighted least-squares training objective that involves learning word vectors so that dot products equal the logarithm of words' probability of co-occurrence. The embeddings are designed so that vector differences capture meaning specified by the juxtaposition of pairs of words, and it is suggested to use Euclidean distance or

cosine similarity between word vectors to produce scalar measures of linguistic or semantic similarity of words. Based on the design and analysis of the embeddings, they are claimed to work well on word analogy tasks [47].

When employing GloVe embeddings, one way to use them is to apply a sequence of token embeddings to a model. Another common way to use GloVe to represent a sequence of tokens is to average the embeddings [6, 63]. This method is effective in capturing general meaning about the words involved, but crucially word-order is lost and so may important semantic information contained in the sequence.

### 5.3.2 Word2Vec Embeddings

Mikolov et al. [42] from Google produced the *Word2Vec* algorithms to produce embeddings for words that encode syntactic and semantic similarities among words. There are two different implementations that produce distributed representations of words, namely the *continuous bag of words* (*CBOW*) and *continuous skip-gram* (*Skip-gram*) models. The processes involved in constructing the Word2Vec embeddings require technically advanced neural network architecture. The machinery behind these algorithms is beyond the scope of this thesis, but we will briefly describe the processes.

The CBOW model tries to predict the current word based on context words. The model get its name because the resulting vectors are averages of vectors and thus word order is ignored. The Skip-gram model, on the other hand, tries to predict surrounding words within a certain range around the current word. The quality of these vectors increases for more surrounding words that are predicted, but this is limited in order to prevent extensive computational complexity requirements.

The Word2Vec algorithms are popular among those studying NLP since it is

very accessible [14]. Sequences of words can be embedded by averaging the word vectors, just like with GloVe. One of the shortcomings with Word2Vec, however, is that it requires a relatively large corpus size for training [3].

### 5.3.3 Doc2Vec Embeddings

Le and Mikolov [37] introduced two *Doc2Vec* algorithms that extend Word2Vec and are a way to represent sequences of words, that is, $n$-grams, sentences, paragraphs, or even entire documents, as vectors. The *Distributed Bag of Words of Paragraph Vectors* (*PV-DBOW* or *DBOW*) operates in the same manner as Skip-gram, but uses a vector representing the word sequence instead and word is order is ignored [36]. The *Distributed Memory Model of Paragraph Vectors* (*PV-DM* or *DM*) works by trying to predict a context word given a vector obtained by concatenating a document vector and word vectors [36].

The paragraph vectors can be used to measure semantic similarity between long pieces of text, as was done for Wikipedia articles [15]. The DBOW model in particular has been used for multi-document summarization [41]. Work done with the Doc2Vec embeddings showed that they worked well in the task duplicate forum question detection when trained on large corpora [36].

## 5.4 String Kernel

The string subsequence kernel was introduced with the motivation to compare text corpora based on their shared substrings [39]. For this kernel, substrings need not be contiguous to contribute to the kernel value, but the degree of the contiguity of a shared substring determines how much weight it will have in the kernel.

### 5.4.1 String Subsequence Kernel

As is done by Lodhi et al. [39], let us define the map that goes into the feature spaces $F_n = \mathbb{R}^{\Sigma^n}$. For a string $s$, the feature mapping $\phi$ is given by defining the $u$ coordinate

$$\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})}, \tag{5.2}$$

for each substring $u \in \Sigma^n$ and a fixed $\lambda \in (0, 1]$. The features counts the number of occurrences of subsequences in the string $s$ and weights them according to their lengths. The contribution of a non-contiguous substring is penalized by the decay factor $\lambda$, which acts as a weight. Thus, the inner product of feature vectors for strings $s$ and $s'$ is the sum over all common subsequences which are weighted according to their frequency of occurrence and their lengths.

**Definition 5.4.1.** [39] The **string subsequence kernel** is defined as

$$\begin{aligned} k_n(s, s') &= \sum_{u \in \Sigma^n} \langle \phi_u(s), \phi_u(s') \rangle = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \sum_{\mathbf{j}:u=s'[\mathbf{j}]} \lambda^{l(\mathbf{j})} \\ &= \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=s'[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}, \end{aligned} \tag{5.3}$$

where $n$ is a pre-specified choice of sequence length.

Given that the definition of the string subsequence kernel does a computation for detecting subsequences for a chosen length $n$, if we want to compute a variation of the string kernel that accounts for all subsequences of lengths up to a chosen $n$, then we must take the sum of these kernel values. This is a valid kernel by Theorem 3.3.1.

A naive implementation of the string subsequence kernel has complexity $O(\Sigma^n)$ [40]. Lodhi et al. [39] proceeded to introduce a recursive procedure that increases the efficiency of computing the subsequence string kernel and reduces the complexity

to $O(n|s||s'|)$. The recursive formulation is provided below, but the format and structure follow the one provided by Beck [6] such that it includes the additional Equation (5.10) that explicitly states a base case of the recursion. The equations are given by

$$k'_0(s, s') = 1 \text{ for all } s \text{ and } s', \tag{5.4}$$

$$\text{for all } i = 1, \ldots, n-1 :$$

$$k''_i(s, s') = 0, \text{ if } \min(|s|, |s'|) < i, \tag{5.5}$$

$$k'_i(s, s') = 0, \text{ if } \min(|s|, |s'|) < i, \tag{5.6}$$

$$k''_i(sa, s'b) = \lambda k''_i(sa, s') \text{ iff } b \neq a, \tag{5.7}$$

$$k''_i(sa, s'a) = \lambda k''_i(sa, s') + \lambda^2 k'_{i-1}(s, s') \text{ otherwise}, \tag{5.8}$$

$$k'_i(sa, s') = \lambda k'_i(s, s') + k''_i(sa, s'), \tag{5.9}$$

$$\text{and finally:}$$

$$k_n(s, s') = 0, \text{if } \min(|s|, |s'|) < n, \tag{5.10}$$

$$k_n(sa, s') = k_n(s, s') + \sum_{j:s'_j=a} \lambda^2 k'_{n-1}(s, s'[1 : j-1]). \tag{5.11}$$

### 5.4.2 Word Sequence Kernel

Cancedda et al. [11] proposed treating words as characters in an alphabet $\Sigma$ and introduced *symbol-dependent decay factors* $\lambda_s$ for each $s \in \Sigma$ that allows for the encoding of semantic similarity between words. Use of symbol-dependent decay factors functions to incorporate prior knowledge about vocabulary into the kernel or to encode information such as their parts-of-speech. For this reason, they call this variation of the string subsequence kernel the *word sequence kernel*. One such way to include additional knowledge with symbol-dependent decay factors is to integrate frequency information obtained with the normalized *inverse documents*

*frequency* (*IDF*) of terms (see Appendix B).

Cancedda et al. [11] further developed the word sequence kernel by adding *independent decay factors for gaps and matches*, $\lambda_g$ and $\lambda_m$.The motivation for making separate decay hyperparameters was so that the it is possible to differentiate the contributions to the kernel value made by contiguous and non-contiguous subsequences. The equations provided here modify the recursive formulation of the string subsequence kernel that were provided previously. The word sequence kernel with independent gap and match decay factors can be obtained by replacing Equations (5.7), (5.8), (5.9), (5.11) with

$$k_i''(sa, s'b) = \lambda_g k_i''(sa, s') \text{ iff } b \neq a, \tag{5.12}$$

$$k_i''(sa, s'b) = \lambda_g k_i''(sa, s') + \lambda_m^2 k_{i-1}'(s, s'), \tag{5.13}$$

$$k_i'(sa, s') = \lambda_g k_i'(s, s') + k_i''(sa, s'), \tag{5.14}$$

$$k_n(sa, s') = k_n(s, s') + \sum_{j:s'_j=a} \lambda_m^2 k_{n-1}'(s, s'[1 : j-1]), \tag{5.15}$$

respectively [6].

In the data analysis portion of this project, we will use the version of the word sequence kernel that uses independent decay factors for gaps and matches but does not include symbol-dependent decay factors. From this point onward, this formulation of the word sequence kernel with independent gap and decay factors will be referenced simply as the *string kernel*.

### 5.4.3 String Kernel

The formulation of the string kernel that we will use is one proposed by Beck [6]. The kernel is a linear combination of string kernels up to a pre-specified choice of

$n$ and is defined as

$$k_{\text{Str}} = \boldsymbol{\mu}^\top \mathbf{k} = \sum_{i=1}^{n} \mu_i k_i, \tag{5.16}$$

where each $k_i$ in $\mathbf{k}$ can be calculated using the recursive formulation previously described and each $\mu_i > 0$ is a distinct weight stored in $\boldsymbol{\mu}$. The vector $\boldsymbol{\mu}$ is a hyperparameter that will be referred to as the *n-gram order coefficients*. This, again, is a valid kernel by Theorem 3.3.1. Use of the string kernel technically does not require the involvement of any embeddings and can operate directly on strings of text. This is the beauty of the string kernel.

Although the string kernel can operate on raw strings, to use them in GPs, it requires that we use real-valued vector representations of the strings. The recursion of the string kernel was unrolled in to a series of for-loops and implemented in a vectorized fashion that uses tensors. The new formulation has complexity $O(n\ell^3)$, where $\ell = \max(|s|, |s'|)$ [6] and requires the use of token-level embedding matrices $E_s$ representing the strings $s$. The final output is the kernel value given by Equation (5.16). First, we define the matrix

$$D_s = \begin{bmatrix} 0 & \lambda_g^0 & \lambda_g^1 & \lambda_g^2 & \dots & \lambda_g^{|s|-2} \\ 0 & 0 & \lambda_g^0 & \lambda_g^1 & \dots & \lambda_g^{|s|-3} \\ 0 & 0 & 0 & \lambda_g^0 & \dots & \lambda_g^{|s|-4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \lambda_g^0 \\ 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}. \tag{5.17}$$

Then, we can proceed to perform the computation in another recursive fashion.

$$S = E_s E_{s'}^\top, \tag{5.18}$$

$$K_1' = \mathbf{1}, \tag{5.19}$$

$$K_\alpha' = D_s^\top K_\alpha'' D_{s'}, \tag{5.20}$$

$$K_\alpha'' = \lambda_m^2 (S \odot K_{\alpha-1}'), \tag{5.21}$$

$$k_1 = \sum_{i,j} S_{ij}, \tag{5.22}$$

$$k_\alpha = \sum_{i,j} \lambda_m^2 (S \odot K_\alpha')_{ij}, \text{ for } \alpha = 2, \ldots, n, \tag{5.23}$$

$$k_{\text{Str}} = \boldsymbol{\mu}^\top \mathbf{k}, \tag{5.24}$$

where $\odot$ is the Hadamard (element-wise) product.

## 5.5 String Kernel with GPyTorch

The string kernel is a rather complex kernel computation. The final version that was used in this project is the vectorized version since we are able to use it with GPs. The string kernel implementation in GPyTorch [24] included the three custom hyperparameters $\lambda_g$, $\lambda_m$, and $\boldsymbol{\mu}$, the last of which is a vector of variable length. GPyTorch has the functionality to optimize the hyperparameters even though gradient functions were not provided for them. This is a benefit when implementing new or complex kernels but has the disadvantage that it adds significant computational time to the GP training.

# Chapter 6

# Model Assessment and Selection

When it comes to model selection, a developer must decide on a method to compare models performing on a given task. There are several different error metrics that can be used to assess performance for regression models.

Let us denote a test set as $\mathcal{D}_*$, where the focus is to get the best model prediction accuracy on this set. If the true targets of the test set are unknown, then $\mathcal{D}_* = \{\mathbf{x}_i : i = 1, ..., N_*\}$. Otherwise, if the true targets are known, then $\mathcal{D}_* = \{(\mathbf{x}_i, y_i) : i = 1, ..., N_*\}$. Let us denote $y_i$ and $\hat{y}_i$ as the true and predicted values of the $i$th target value, respectively, where these can come from either the training or test set. In the context of GPs, for example, $\hat{y}_i$ could be the GP posterior mean $\bar{f}$ as in Equation (2.22). Let us also denote a residual as $e_i = y_i - \hat{y}_i$ which is the difference between the $i$th observed and predicted values [31].

## 6.1 Error Metrics

When assessing model performance, there are scale-dependent and correlation measures that are commonly used to measure accuracy on a test set where the true

target values are known.

### 6.1.1 Scale-dependent Measures

In regression, the most commonly-used error metric is the *mean squared error* (*MSE*) [31] given by

$$\text{MSE} = \frac{1}{N_*}\sum_{i=1}^{N_*}(e_i)^2. \tag{6.1}$$

Other commonly used variations of this metric are the *root mean squared error* (*RMSE*) computed as [30]

$$\text{RMSE} = \sqrt{\text{MSE}}. \tag{6.2}$$

and the *mean absolute error* (*MAE*) computed as [30]

$$\text{MAE} = \frac{1}{N_*}\sum_{i=1}^{N_*}|e_i|. \tag{6.3}$$

The RMSE can be preferred over MSE because it is more interpretable since it is on the same scale of the data [30], like the MAE. One important observation to make, though, is that, in comparison to MAE, the RMSE penalizes more heavily for poorly predicted values [12]. Moreover, the MAE metric is suitable to describe uniformly distributed errors, but the underlying assumption for the RMSE is that the errors are unbiased and follow a normal distribution [12]. One can use any of these error metrics, but the comparison across different error metrics produced by a single model is not meaningful as they are on different scales or have different assumptions. Therefore, comparison of error metrics across selected models should be done metric-wise in order to get meaningful results.

### 6.1.2 Correlation Measures

The *Pearson's product-moment correlation coefficient* $r$ is a measure of any linear trend between two sets of values [48]. If the true target values of the test set are known, then this can be used on the pairs of true and predicted values $(y_1, \hat{y}_1), \ldots, (y_{N_*}, \hat{y}_{N_*})$ to assess prediction accuracy. The Pearson's $r$ correlation coefficient is defined as

$$r = \frac{\sum_{i=1}^{N_*}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{N_*}(y_i - \bar{y})^2 \ \sum_{i=1}^{N_*}(\hat{y}_i - \bar{\hat{y}})^2}}, \tag{6.4}$$

where $\bar{y}$ and $\bar{\hat{y}}$ are sample means. Since this metric is a measure of correlation that is bounded to $[-1, 1]$, the goal is to achieve the greatest value of $r$ possible as it indicates the highest degree of positive correlation between predicted and true target values.

## 6.2 Resampling Methods

A *resampling method* is a statistical tool which involves repeatedly drawing samples from a training set and refitting a model of interest to gain insight about the fitted model [31]. Resampling methods estimate the test set error rate by holding out a subset of the training observations from the fitting process and then applying the model of interest to the held out observations [31]. In cases where the true target values of the test set $\mathcal{D}_*$ are not known, it can be useful to estimate model performance using resampling methods applied to the training set $\mathcal{D}$. This will allow the model developer to have an idea of how well the model will do during final prediction on the test set.

### 6.2.1 Validation Sets

One approach that is used is the *validation set approach* in which a subset of the training set is held out and designated as a *validation set*. The retained training set is subsequently used to train the model, and performance is assessed on the validation set, which effectively acts like a test set where the true target values are known. For example, one way execute this is to randomly designate 30% of the original training observations as a validation set and use the other 70% as the training set. In this way, the model performance can be assessed using the error metrics described in Section 6.1. The validation set approach is useful in that it simulates a situation where the true values of a test set are known.

One drawback to this approach is that the model will be subjected to variability in the validation set error metrics that will depend on which observations are included the training and validations sets [31]. Another drawback is that the model is fitted on a subset of the training data, meaning that it is not fitting on all available training data. Since models tend to perform worse when trained on fewer observations, the validation set may tend to overestimate the test error metrics for the model fit on the whole training set [31]. In practice, when a best model is obtained using this approach, this model is refitted on all the data before being used to predict on the test set.

### 6.2.2 $k$-Fold Cross-Validation

*$k$-fold cross-validation* ($CV$) is a method that applies to the training set in which we repeat the validation set approach $k$ times. This approach entails randomly dividing the set of training observations into $k$ groups, or *folds*, of approximately equal size [31]. The validation approach is then applied with each fold treated as

a validation set. This process produces $k$ times the number of error metrics of interest. We typically take the mean of each of the error metrics obtained in the CV to make the metrics comparable across models. Variability in error metrics produced by performing CV will typically be lower than the variability of the error metrics obtained from the simple validation set approach [31].

For this process to work, $k$ must be an integer that satisfies $1 \leq k \leq N$. In practice, $k$ is usually chosen to be 5, 10, or $N$ [31], with the last being a special case known as *leave-one-out cross-validation* (*LOO-CV*). Fitting a model has the potential to be computationally expensive and plays a role in the decision for choosing a smaller $k$. The other factor that plays a role in the choice of $k$ is a matter of the bias-variance trade-off [31].

When performing CV, the bias-variance trade-off comes into play when we consider the *expected prediction error* for a regression fit at an input point, which is the sum of the *irreducible error*, the *squared bias*, and the *variance* [28]. The irreducible error is the variance of the target around its true mean and is unavoidable [28], the squared bias is the amount by which the average of the estimate differs from the true mean [28], and the variance is the amount a prediction would change if we used a different training set [31]. If we choose $k = N$, then the LOO-CV will produce an expected prediction error that will be nearly unbiased since each fold operates on $N - 1$ training points at a time, almost the whole training set. The variance in the expected prediction error, however, will be greater since we are effectively averaging the outputs of $N$ fitted models, each with a nearly identical set of observations and thus highly positively correlated outputs [31]. On the other hand, if we perform a $k$-fold cross-validation with $k < N$, then the expected prediction error for the regression fit will have more bias since we are training on a smaller sample

of the training data. Moreover, this situation would yield us averaging the outputs of $k$ fitted models that are less correlated with each other since there is less overlap between the training sets constructed by holding out each fold, resulting in lower variance in the expected prediction error [31]. Thus, the takeaway is that there is more bias and less variance for a smaller choice of $k$, and there is less bias but more variance for a larger choice of $k$.

## 6.3 LOO-CV Error Metrics for GPs

Since LOO-CV entails performing a validation approach $N$ times, this can potentially be an expensive procedure to implement. However, when using LOO-CV to assess the performance of a GPR model, the model need not be fitted $N$ times. For a GP with mean zero, the predictive mean expression is

$$\mu_i = y_i - [K_{\mathbf{y}}^{-1}\mathbf{y}]_i / [K_{\mathbf{y}}^{-1}]_{ii} \tag{6.5}$$

and the predictive variance expression is

$$\sigma_i^2 = 1/[K_{\mathbf{y}}^{-1}]_{ii}, \tag{6.6}$$

where $K_{\mathbf{y}}$ is the covariance matrix of $\mathbf{y}$ associated with the GP, as given in Equation 2.17 [49, sec. 5.4.2]. Thus, LOO-CV allows for the evaluation of models with the efficient computation of CV predictive means that can be achieved with a little more than a matrix inversion.

In conjunction with the LOO-CV predictive mean given by Equation (6.5), we can use the known target values $y_i$ in our training set to compute LOO-CV scale-dependent error metrics. In this manner, $\mu_i$ acts as our $\hat{y}_i$.

## 6.4 Forward Stepwise Selection

*Forward stepwise selection* is a systematic process in which we choose subsets of variables by adding one variable at a time to a previously chosen subset [50]. This model selection process is a bottom-up approach in which we iteratively complicate models until it fails to improve a chosen metric by a specified error loss. This process requires that the user selects an error metric to compare models produced during each iteration. Moreover, a termination rule should be set, otherwise forward selection continues until the most complicated model is constructed [50].

The motivation for employing this process is to avoid computing the regression for all possible models. Testing all possible models is generally computationally expensive and probably inefficient. One of the hopes when using forward selection is that there is a clear connection between high-performing models such that they share one-step complications. In this case, it is highly likely that the final model produced is the best-performing model. This outcome is not guaranteed, however, as it could very well be the case with models that have lots of variables or parameters that a best-performing model can be isolated or disconnected from other high-performing models by one-step complications. It is situations like these where the best-performing model will go undiscovered by this process.

# Chapter 7

# Data Analysis of Affective Text

The data that we will analyze is the *SemEval 2007 Affective Text Dataset* produced by Carlo Strapparava and Rada Mihalcea [59]. The data originates from two datasets: a development dataset with 250 annotated headlines and a test dataset with 1000 annotated headlines. Each annotation is a vector containing six emotions scores and a valence score, where the emotions that were scored are anger, disgust, fear, joy, sadness, and surprise.

## 7.1 The Data

The test dataset was independently labeled by six untrained annotators that were instructed to annotate based on the presence of words or phrases conveying emotional content so that the focus was on emotional lexical semantics [59]. A corpus extracted from *WordNet Affect* [60] with words associated to the emotions of interest was provided, but participants had the option to use this or any resources they wanted.

The headlines are news titles extracted from newspapers or news websites such

as New York Times, CNN, BBC News, or Google News. Headlines were chosen because they are designed to provoke emotional reactions to attract readers and are close to sentences in structure.

All of the scores are integer values. All of the emotions are on a scale of 0 to 100, where 0 indicates that the emotion is not evoked and 100 implies "maximum emotional load" [59]. The valence score is on a scale of -100 to 100, where -100 and 100 correspond to strong negative and positive emotional reactions, respectively. The use of such scales was motivated by the intention of using finer-grained scales than simply 0/1 annotations for emotions and -1/0/1 annotations for polarity orientations.

### 7.1.1 Observations and Speculations

Some observations can be made about the training set. A coarse-grained examination of the training set is provided in Table 7.2. It is apparent that in the training set, there is a significant tendency to not have an emotion expressed based on the number of zeros present in the emotion scores. It can also be extrapolated that in general, it is not probable to express an emotion strongly. This is to say that it is not very common for an emotion to exceed a score 50, much less to score above a 75, according to the data in the training set. This could play a role in biasing our models towards predicting relatively smaller scores in the test set than what may be appropriate, but this remains to be determined.

Another observation is that a significant proportion (about 70%) of the valence scores were reported as neutral with scores in the interval $[-50, 50]$. This makes sense in relation to the emotion scores since the majority of the emotion were scored 50 or less across the board. Moreover, it is the case that there were more negative

Table 7.1: Sample of the training dataset containing the headlines with the associated emotion and valence scores. The columns correspond to the scores associated with anger, disgust, fear, joy, sadness, surprise, and valence, in that order.

| Headline | Ang | Dis | Fear | Joy | Sad | Sur | Val |
|---|---|---|---|---|---|---|---|
| Sony apologises for global battery recall | 6 | 0 | 0 | 0 | 29 | 0 | -16 |
| Dow hits new record, eyes 12,000 | 0 | 0 | 0 | 49 | 0 | 0 | 40 |
| Advantage Cardinals with Suppan in Game 4 | 0 | 0 | 0 | 4 | 0 | 0 | 2 |
| No evidence of ice reserves on the moon | 0 | 0 | 0 | 0 | 16 | 10 | -12 |
| Iraqi Journalists Add Laws to List of Dangers | 7 | 0 | 47 | 0 | 28 | 13 | -63 |
| Amish schoolhouse torn down | 29 | 27 | 37 | 0 | 73 | 2 | -56 |
| Nigeria hostage feared dead is freed | 18 | 0 | 52 | 66 | 20 | 65 | 31 |

Table 7.2: Coarse-grained examination of the observations from the training set. The closed sets in the rows labelled "Output" correspond to the sets to which the respective scores are a part of.

| Output | $\{0\}$ | $(0, 25]$ | $(25, 50]$ | $(50, 75]$ | $(75, 100]$ |
|---|---|---|---|---|---|
| Anger | 113 | 76 | 44 | 17 | 0 |
| Disgust | 144 | 54 | 43 | 9 | 0 |
| Fear | 122 | 57 | 42 | 24 | 5 |
| Joy | 118 | 58 | 40 | 22 | 12 |
| Sadness | 96 | 53 | 62 | 28 | 11 |
| Surprise | 105 | 104 | 33 | 8 | 0 |
| Output | $[-100, 50)$ | $[-50, 0)$ | $\{0\}$ | $(0, 50]$ | $(50, 100]$ |
| Valence | 55 | 93 | 9 | 74 | 19 |

valence scores with a proportion of just under 60%.

Further investigation into the development set revealed that there are four instances in which the scores associated with the headline are all zeros. It is unexpected that there are any instances at all that did not signal emotions or valence in any capacity.

Analytically speaking, for all emotion scores, the score is relatable to the magnitude of the associated emotion, where it is bounded below by 0 and bounded above by 100. The same applies to the valence score, except it applies to the absolute value. This doubled range for valence scores in comparison to the emotion scores will make comparing error metrics across models predicting emotion and valence suffer from not being directly comparable.

Joy is the only emotion that can certainly be associated with a positive valence score. Anger, disgust, fear, and sadness comprise four of the six emotions being scored in this dataset, and each of these emotions is inherently associated with negative valence. In line with the statement made by Ectoff and Magee (in press) [18], surprise is the only emotion that is not necessarily associated with a positive or negative reaction, however, it remains to be seen if there is any association between surprise and valence scores.

The emotions being studied are clearly distinct feelings, but it clear that there are some degrees of correlation (or anti-correlation) between the different emotions. For example, joy and sadness are conventionally polar opposites and share the relation that more of one intuitively implies less of the other. The valence score hypothetically will be correlated most with the joy and sadness emotions.

Inspection of the correlation matrix between scores in the training set indeed shows that valence and joy share the highest degree of correlation and valence and

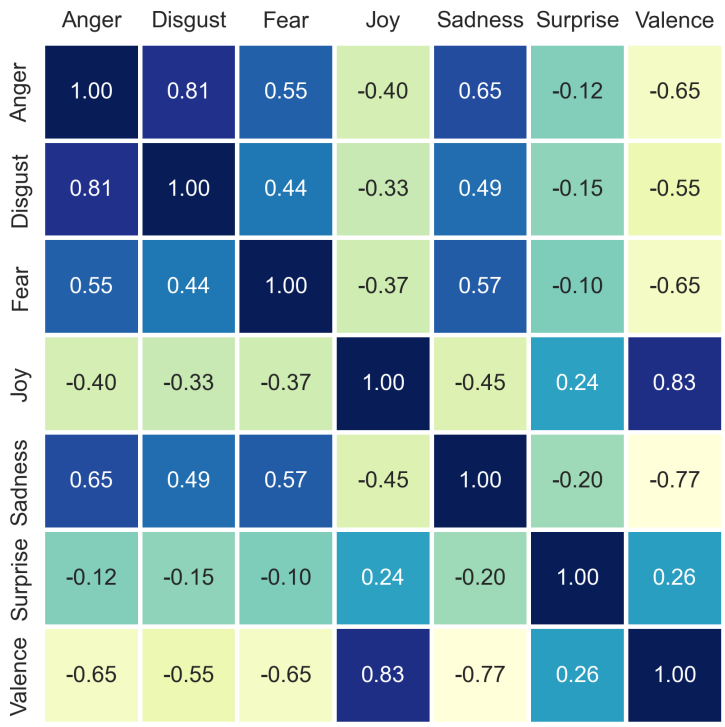


Figure 7.1: Correlation matrix for the emotion and valence scores in the training dataset. Greater (darker) values indicates more correlation between values, and lesser (lighter) values indicate more anti-correlation between values.

sadness share the highest degree of anti-correlation. The anger and disgust emotions show the next highest degree of positive correlation, followed by anger and sadness, fear and sadness, and anger and fear. All of these are commonly associated with negative feelings, so it makes sense why these are positively correlated. Finally, the emotion sharing the least correlation with valence is surprise, but the emotions sharing the highest degrees of correlation with surprise are joy and sadness. In our data analysis, we will have the opportunity to inspect our models to determine if this was something that could be detected in the data.

### 7.1.2 Motivation

We will use the Affective Text dataset to conduct emotion and sentiment analyses, in particular, to predict average scores of affect. We will use the development set of size 250 for training, and we will treat the test set of size 1000 as a validation set. We will be modeling with GPs and will be using a specified set of selected embeddings and kernels to determine if we can accurately predict emotion and valence scores associated with the news headlines. Then, we can perform an analysis on the performance of our models.

The order in which the models are developed is intended to illustrate a progression of complexity in terms of how the models can be configured. The speculation is that there will be notable improvement as the models become increasingly complex.

## 7.2 Data Preprocessing

We will treat $Anger$, $Disgust$, $Fear$, $Joy$, $Sadness$, $Surprise$, and $Valence$ as variables during our analysis. We will start by developing a notation to represent

the vector of scores. Let

$$y_i = (Ang, Dis, Fear, Joy, Sad, Sur, Val). \tag{7.1}$$

In the interest of using normalized data for the GPs, the scores were pre-transformed by dividing all scores by 100. The consequence of this was rescaled data in $[0, 1]$ and $[-1, 1]$ for the emotion and valence scores, respectively.

A GP requires real-valued vector inputs $\mathbf{x}$. Therefore, the inputs will be constructed by storing the concatenated embeddings into vectors. For now, let $\mathcal{D}_s = \{(s_i, y_i) : i = 1, \ldots, N\}$ and $\mathcal{D}_{s*} = \{(s_i, y_i) : i = 1, \ldots, N_*\}$ denote the raw training and test datasets containing the news headlines in the form of strings and the annotated scores. Since the input data are news headlines composed of strings of characters, it called for preprocessing the data.

**Stage 1**: First and foremost, quotes (single and double) around headlines were removed. Next, instances of doubled-double quotes (`"" ""`) were reduced to single double quotes (`" "`). Then, one-off characters were removed: a tab character was replaced with a space character (as was intended for its original use) and a grave accent character was replaced with a single quote (as it was used erroneously). The headlines were then downcased to reduce the number of characters. The character-level one-hot encoding and GloVe embeddings used the headlines in this manner.

**Stage 2**: For the word-level one-hot encoding and Doc2Vec embeddings, the headlines were refined further. First, all hyphens were replaced with spaces so that more distinct words were created in instances containing hyphenated words. The motivation for this was to reduce the number of single instances of hyphenated compound words which were unlikely to appear more than once. Then, all punctuation was removed from the headlines, leaving only letters, numbers, and spaces.

## 7.3 Embedding Vectors

### Character-level One-hot Encoding

The character-level encoding scheme is relatively straightforward once the data was preprocessed. The embeddings must be in the form of an embedding matrix for our purposes.

### Word-level One-hot Encoding

The word-level encoding required choosing a vocabulary size to run the algorithm to create the embeddings. The vocabulary size that was chosen was 5000 to overestimate the number of unique words in the aggregated dataset totaling 3438 after the second preprocessing stage. The MD-5 hash function was used for reproducibility. These embeddings must also be in the form of an embedding matrix for our purposes.

### GloVe

The GloVe dimension that was chosen was 50. Based on some preliminary experiments not discussed in this thesis, it was empirically determined that there was only modest improvement by choosing a larger sized embedding. When the strings were tokenized, if any of the tokens were not present in the GloVe dataset, then the tokens were simply disregarded and not incorporated into the embedding.

There are two ways that GloVe embeddings were used in our analysis. For the first method, each token was extracted from the string and then the vectors were stacked to create an embedding matrix. The other method goes a step further and averages the embedding matrix to result in 50-dimensional vector.

### Doc2Vec

Doc2Vec had several parameters to specify upon creating the embeddings. First and foremost, the seed (for the random number generator) and number of workers (number of worker threads to train the model) were both set to 1 for reproducibility. Next, the window size was chosen to be 4, which is the maximum distance between the current and predicted word within a sentence. This choice was made since the the minimum, maximum, and mean number of words in the aggregate dataset after Stage 2 of preprocessing was 2, 15, and approximately 6.57, respectively. Finally, the dimensionality for the Doc2Vec feature vectors was also chosen to be 4 for both the DM and DBOW algorithms, which was determined to work well in some preliminary experiments.

## 7.3.1 Concatenated Embedding Vectors

In our analysis, each vector $\mathbf{x}$ will denote the concatenated list of embeddings representing the news headlines. These vectors will allow for the construction of our datasets $\mathcal{D}$ and $\mathcal{D}_*$. The embeddings will be assigned codes to ease notation: character-level one-hot encoding (T1), word-level one-hot encoding (T2), token-level GloVe (T3), averaged GloVe (S1), Doc2Vec DM (S2), and Doc2Vec DBOW (S3). The code "T" was given to the embeddings that get stored in a flattened embedding matrix and operate on a *token* level. Likewise, the code "S" was given to the remaining embeddings that operate on distributed representations of words, or *sequences* of tokens.

Now, we may formulate the input vector $\mathbf{x}$ which will be given by

$$\mathbf{x} = (\mathbf{x}_{\mathrm{T1}}, \mathbf{x}_{\mathrm{T2}}, \mathbf{x}_{\mathrm{T3}}, \mathbf{x}_{\mathrm{S1}}, \mathbf{x}_{\mathrm{S2}}, \mathbf{x}_{\mathrm{S3}}). \tag{7.2}$$

Table 7.3: Codes and corresponding embeddings.

| Code | Embedding |
|---|---|
| T1 | character-level one-hot encoding |
| T2 | word-level one-hot encoding |
| T3 | token-level GloVe |
| S1 | averaged GloVe |
| S2 | Doc2Vec DM |
| S3 | Doc2Vec DBOW |

The dimensionality of $\mathbf{x}$, which we will call $\omega$, depends on the dimensionalities of its component vectors such that

$$\omega = \omega_{\mathrm{T1}} + \omega_{\mathrm{T2}} + \omega_{\mathrm{T3}} + \omega_{\mathrm{S1}} + \omega_{\mathrm{S2}} + \omega_{\mathrm{S3}}. \tag{7.3}$$

The dimensionalities for the sequential embeddings $\omega_{\mathrm{S1}}$, $\omega_{\mathrm{S2}}$, and $\omega_{\mathrm{S3}}$ depend on the user's choice of embeddings sizes for GloVe and Doc2Vec. The dimensionalities for the other embeddings, however, require more work.

The dimensionality $\omega_{\mathrm{T1}}$ is given by the alphabet size times the maximum length of all the strings in the dataset. The dimensionality $\omega_{\mathrm{T2}}$ is given by the vocabulary size times the maximum number of words in all of the strings. The dimensionality $\omega_{\mathrm{T3}}$ is given by the selected GloVe embedding size times the maximum number of tokens. In this way, all of the strings can be embedded in a vector $\mathbf{x}$. For the cases where the maximum number of characters, words, or tokens is not reached, as is with most cases, the simplest solution is to pad the embedding vectors with trailing zeros so that the vectors from each embedding have the same dimensionality prior to concatenation.

A suitable way to approach this in practice is to impose constraints on the accepted string data such that the alphabet size, maximum number of characters, vocabulary size, maximum number of words, GloVe embedding size, and maximum number of GloVe tokens are pre-specified. In a situation such as this one where the training and test inputs are known, the user should combine the datasets and proceed to determine these values empirically. In doing this, the token-level embedding dimensions can be determined for $\mathbf{x}_{\mathrm{T}1}$, $\mathbf{x}_{\mathrm{T}2}$, and $\mathbf{x}_{\mathrm{T}3}$.

## 7.4 Custom Kernels

We will use a selection of different kernels that are equipped with the embeddings just described. We will be using the string, linear, polynomial, radial basis function, and Matérn kernels. The polynomial kernel will be restricted to degree $q = 2$. The Matérn kernels will be varied such that the smoothness hyperparameter $\nu$ will be set to 0.5, 1.5, and 2.5. The string kernel in particular will use the character-level one-hot encoding (T1), word-level one-hot encoding (T2), and the token-level GloVe (T3). All other kernels will be used with the averaged GloVe (S1), Doc2Vec DM (S2), and Doc2Vec DBOW (S3) embeddings.

We will now introduce custom kernels that use the embeddings listed above and choose a set of dimensions to operate on. The string kernel that we will use will be defined as

$$k_{\mathrm{Str},\mathrm{T}i}(\mathbf{x}, \mathbf{x}') = k_{\mathrm{Str}}(\mathbf{x}_{\mathrm{T}i}, \mathbf{x}'_{\mathrm{T}i}), \tag{7.4}$$

where $i = 1, 2, 3$ specifies the token-level embedding being employed. The others

follow suit:

$$k_{1,\mathrm{S}i}(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}_{\mathrm{S}i}, \mathbf{x}'_{\mathrm{S}i}), \tag{7.5}$$

$$k_{2,\mathrm{S}i}(\mathbf{x}, \mathbf{x}') = k_2(\mathbf{x}_{\mathrm{S}i}, \mathbf{x}'_{\mathrm{S}i}), \tag{7.6}$$

$$k_{\mathrm{RBF},\mathrm{S}i}(\mathbf{x}, \mathbf{x}') = k_{\mathrm{RBF}}(\mathbf{x}_{\mathrm{S}i}, \mathbf{x}'_{\mathrm{S}i}), \tag{7.7}$$

$$k_{\nu=1/2,\mathrm{S}i}(\mathbf{x}, \mathbf{x}') = k_{\nu=1/2}(\mathbf{x}_{\mathrm{S}i}, \mathbf{x}'_{\mathrm{S}i}), \tag{7.8}$$

$$k_{\nu=3/2,\mathrm{S}i}(\mathbf{x}, \mathbf{x}') = k_{\nu=3/2}(\mathbf{x}_{\mathrm{S}i}, \mathbf{x}'_{\mathrm{S}i}), \tag{7.9}$$

$$k_{\nu=5/2,\mathrm{S}i}(\mathbf{x}, \mathbf{x}') = k_{\nu=5/2}(\mathbf{x}_{\mathrm{S}i}, \mathbf{x}'_{\mathrm{S}i}), \tag{7.10}$$

where $i = 1, 2, 3$ specifies the sequence-level embedding being employed.

To represent the kernels more concisely, we will use the following shorthands to ease notation: string (STR), linear (LIN), polynomial (POLY), radial basis function (RBF), and Matérn (MAT) followed by hyperparameter specification of $\nu$. We will follow these with the embeddings being used in the kernel. From here on out, the set of kernels that we will be using and referencing is

$$\begin{aligned}
&\{\text{STR[T1]}, \quad \text{STR[T2]}, \quad \text{STR[T3]}, \\
&\ \text{LIN[S1]}, \quad \text{LIN[S2]}, \quad \text{LIN[S3]}, \\
&\ \text{POLY[S1]}, \quad \text{POLY[S2]}, \quad \text{POLY[S3]}, \\
&\ \text{RBF[S1]}, \quad \text{RBF[S2]}, \quad \text{RBF[S3]}, \\
&\ \text{MAT(0.5)[S1]}, \quad \text{MAT(0.5)[S2]}, \quad \text{MAT(0.5)[S3]}, \\
&\ \text{MAT(1.5)[S1]}, \quad \text{MAT(1.5)[S2]}, \quad \text{MAT(1.5)[S3]}, \\
&\ \text{MAT(2.5)[S1]}, \quad \text{MAT(2.5)[S2]}, \quad \text{MAT(2.5)[S3]} \ \}.
\end{aligned} \tag{7.11}$$

We will refer to the kernels in the set 7.11 as *basic* kernels because they will eventually serve as the building blocks for compound kernels that are constructed later.

It should be noted that values of $n$ for the $n$-gram order coefficients are pre-

specified for the string kernels. STR[T1], STR[T2], and STR[T3] were specified such that the choices of $n$ for the $n$-gram coefficients were set to 4, 6, and 3, respectively. It was necessary to choose small values because of computational limitations. The motivation for selecting these three values comes from the following reasoning. (1) Character-level one-hot encoding is used for detecting patterns in sequences of characters. The characters themselves carry little semantic information, and since there were less than 90 total characters that were encoded with this scheme, it was decided that examining subsequences up to length 4 across the whole string was sufficient for the recognition of patterns among substring sequences. (2) Since the number of words in our aggregate dataset approached 3500 distinct words (after Stage 2 of preprocessing), it was decided that contiguous sequences of at least 6 would be able to capture semantic information. (3) The GloVe embedding incorporates distributed representations of words in each token within the string. For this reason, a smaller choice of $n$ was decided to be sufficient. Given how short most of the headlines are, the $n$-gram order was selected to be 3.

When we begin to compound kernels by means of kernel addition or multiplication, we will use the intuitive format of connecting them by the symbols + and $\times$, respectively. For the multi-task cases, we will include the rank by following the kernel with a hyphen and then the rank number which we will denote $\rho$. Last but not least, when we have LMC kernels, we will reuse the symbol + to combine component kernels.

As a quick aside, when we initialize our GPs in GPyTorch, one assumption that was made is that the output data have a zero mean. Let $\tau$ be the number of training iterations that are used to optimize the hyperparameters of a GP. Finally, in our analysis, we will resort to referring to the GP equipped with a kernel by just

naming the kernel to avoid redundancy.

## 7.5 Independent Single-task GPs

The simplest approach that can be implemented in terms of model complexity is to create an independent GP for each task. The approach will be to cycle through the kernels in the set 7.11 to see how the models perform on a task-by-task basis.

### 7.5.1 String Kernels

We found in a preliminary ModelList trial that it is appropriate to let $\tau = 100$ (see Appendix C). However, the string kernel received special treatment in that the GPs were trained with $\tau = 50$ because of the computational requirements to train the GPs.

Recall that there were three different variations of the string kernel upon instantiation. STR[T1], STR[T2], and STR[T3] were varied such that the values of $n$ for the $n$-gram coefficients were set to 4, 6, and 3, respectively. Moreover, with the intention of reproducibility, the hyperparameters for all of these kernels were initialized to $\lambda_g = 1/2$, $\lambda_m = 1/2$, and $\boldsymbol{\mu} = \mathbf{1}$.

#### Results

The results for this analysis are displayed in Table 7.4. The data show that the string kernel with the GloVe embedding outperforms the others. Technically, the test set RMSE for STR[T2] is lower for the disgust task, but the difference is marginal. Despite STR[T3] performing generally better than the others, there were no RMSE metrics below 10, and there were no Pearson $r$ values exceeding 0.70.

Table 7.4: LOO-CV RMSE, test set RMSE, and test set Pearson $r$ metrics for the single-task GPs equipped with the string kernels. The columns represent the metrics for anger, disgust, fear, joy, sadness, surprise, and valence, respectively.

| Kernel | Ang | Dis | Fear | Joy | Sad | Sur | Val |
|---|---|---|---|---|---|---|---|
| | | | LOO-CV RMSE | | | | |
| STR[T1] | 18.59 | 16.43 | 23.00 | 26.13 | 25.30 | 15.11 | 42.04 |
| STR[T2] | 16.67 | 15.18 | 21.86 | 25.10 | 22.08 | 16.30 | 38.39 |
| STR[T3] | **15.79** | **14.46** | **18.55** | **22.79** | **20.37** | **15.07** | **33.18** |
| | | | Test Set RMSE | | | | |
| STR[T1] | 13.66 | 10.82 | 19.55 | 21.81 | 22.46 | 16.38 | 48.01 |
| STR[T2] | 13.58 | **10.46** | 20.36 | 24.51 | 22.82 | 19.94 | 46.22 |
| STR[T3] | **12.73** | 10.89 | **15.95** | **20.67** | **19.78** | **16.18** | **40.28** |
| | | | Test Set Pearson $r$ | | | | |
| STR[T1] | 0.23 | 0.15 | 0.29 | 0.01 | 0.18 | 0.12 | 0.22 |
| STR[T2] | 0.29 | 0.22 | 0.35 | 0.08 | 0.29 | 0.15 | 0.35 |
| STR[T3] | **0.49** | **0.34** | **0.64** | **0.36** | **0.52** | **0.28** | **0.57** |

This is to say that performance was not poor, but it was not amazing either.

The LOO-CV RMSE metrics tended to overestimate the test set RMSE metrics, other than the surprise and valence tasks where it underestimated. The LOO-CV RMSE metrics suggest that STR[T1] models have comparable performance to the STR[T2] models, other than perhaps the sadness and valence tasks. The test set RMSE metrics also suggest that the models have comparable performance, where the STR[T1] models do somewhat better in the joy and surprise tasks. In general, the Pearson $r$ values suggest that all models with STR[T2] outperform the models with STR[T1].

It appears that detecting joy was the *hardest* task given that it has some of the highest RMSE metrics and relatively low Pearson $r$ coefficients. On the other hand, the RMSE metrics would suggest that the detecting disgust was the *easiest* task, while the Pearson $r$ coefficients suggest that fear was the *easiest* to detect. This discrepancy could be explained by variability of the scores for the emotions, but it would require examining the test set scores to know for certain. The valence task fared pretty well in that it had RMSE metrics that were not very large, considering that it has double the range to predict on in comparison to the emotion scores. Moreover, the Pearson $r$ coefficients suggests that the models were able to perform well on this task relative to the others.

### 7.5.2 Basic Kernels

We are now going to analyze the performance of GPs equipped with the rest of the kernels in the set 7.11. The computational time needed to perform the training for these GPs is exceptionally low compared to the string kernel and much more manageable. Thus, we will now fix $\tau = 100$.

Table 7.5: LOO-CV RMSE metrics for the independent single-task GPs. *Light-to-dark* was applied to all the GPs at once but was applied to the set of the emotion scores and valence scores separately. Smaller (lighter) scores are better.

| | Anger | Disgust | Fear | Joy | Sadness | Surprise | Valence |
|---|---|---|---|---|---|---|---|
| LIN[S1] | 14.95 | 13.87 | 18.49 | 21.54 | 19.43 | 15.19 | 33.12 |
| LIN[S2] | 18.66 | 16.67 | 23.61 | 25.75 | 25.10 | 15.43 | 42.73 |
| LIN[S3] | 18.66 | 16.66 | 23.61 | 25.74 | 25.10 | 15.43 | 42.73 |
| POLY[S1] | 15.22 | 13.55 | 19.00 | 21.21 | 19.63 | 15.86 | 32.60 |
| POLY[S2] | 18.96 | 16.92 | 23.97 | 25.80 | 25.63 | 15.48 | 42.96 |
| POLY[S3] | 18.96 | 16.91 | 23.95 | 25.84 | 25.64 | 15.47 | 42.97 |
| RBF[S1] | 14.92 | 13.75 | 18.29 | 21.06 | 19.09 | 15.05 | 32.31 |
| RBF[S2] | 18.62 | 16.63 | 23.56 | 25.71 | 25.08 | 15.39 | 42.68 |
| RBF[S3] | 18.62 | 16.63 | 23.56 | 25.70 | 25.08 | 15.39 | 42.67 |
| MAT(0.5)[S1] | 14.25 | 13.18 | 18.23 | 19.88 | 18.31 | 14.66 | 30.96 |
| MAT(0.5)[S2] | 18.76 | 16.75 | 23.74 | 25.65 | 25.23 | 15.41 | 42.85 |
| MAT(0.5)[S3] | 18.76 | 16.74 | 23.74 | 25.65 | 25.24 | 15.39 | 42.84 |
| MAT(1.5)[S1] | 14.74 | 13.49 | 18.25 | 20.26 | 18.68 | 14.92 | 31.46 |
| MAT(1.5)[S2] | 18.67 | 16.68 | 23.62 | 25.69 | 25.13 | 15.41 | 42.75 |
| MAT(1.5)[S3] | 18.67 | 16.67 | 23.61 | 25.69 | 25.14 | 15.40 | 42.74 |
| MAT(2.5)[S1] | 14.83 | 13.61 | 18.26 | 20.51 | 18.84 | 14.99 | 31.72 |
| MAT(2.5)[S2] | 18.65 | 16.66 | 23.59 | 25.71 | 25.11 | 15.41 | 42.72 |
| MAT(2.5)[S3] | 18.65 | 16.65 | 23.59 | 25.70 | 25.11 | 15.41 | 42.72 |

### Results

The results obtained for the LOO-CV RMSE metrics can be found in Table 7.5. There is evidence of lighter coloration of the table in every third row, starting from the first one. This implies that the kernels using the GloVe embeddings have an edge over kernels using the Doc2Vec embeddings. The kernel MAT(0.5)[S1] outperformed all the others across all LOO-CV RMSE metrics, however it is not by a significant amount. Nonetheless, the metrics suggest that this will be the superior

Table 7.6: Test set RMSE metrics for the independent single-task GPs. *Light-to-dark* was applied to all the GPs at once but was applied to the set of the emotion scores and valence scores separately. Smaller (lighter) scores are better.

| | Anger | Disgust | Fear | Joy | Sadness | Surprise | Valence |
|---|---|---|---|---|---|---|---|
| LIN[S1] | 13.15 | 11.26 | 15.80 | 19.35 | 19.42 | 15.82 | 40.48 |
| LIN[S2] | 14.44 | 11.20 | 20.45 | 21.19 | 23.02 | 15.97 | 48.90 |
| LIN[S3] | 14.44 | 11.20 | 20.44 | 21.19 | 23.02 | 15.97 | 48.89 |
| POLY[S1] | 13.74 | 12.08 | 15.92 | 20.67 | 20.07 | 17.30 | 40.47 |
| POLY[S2] | 14.48 | 11.11 | 20.62 | 21.96 | 22.99 | 16.38 | 49.03 |
| POLY[S3] | 14.49 | 11.13 | 20.66 | 21.90 | 22.99 | 16.39 | 49.03 |
| RBF[S1] | 12.93 | 11.12 | 15.45 | 19.45 | 19.05 | 15.63 | 40.02 |
| RBF[S2] | 14.44 | 11.20 | 20.42 | 21.16 | 22.98 | 15.94 | 48.92 |
| RBF[S3] | 14.43 | 11.20 | 20.42 | 21.16 | 22.97 | 15.94 | 48.91 |
| MAT(0.5)[S1] | 12.68 | 11.02 | 15.20 | 18.89 | 18.87 | 15.67 | 39.88 |
| MAT(0.5)[S2] | 14.49 | 11.28 | 20.48 | 21.46 | 23.04 | 15.98 | 49.03 |
| MAT(0.5)[S3] | 14.48 | 11.28 | 20.48 | 21.45 | 23.02 | 15.99 | 49.02 |
| MAT(1.5)[S1] | 12.81 | 11.12 | 15.26 | 19.28 | 18.96 | 15.65 | 39.96 |
| MAT(1.5)[S2] | 14.45 | 11.24 | 20.44 | 21.26 | 22.99 | 15.92 | 48.95 |
| MAT(1.5)[S3] | 14.44 | 11.24 | 20.44 | 21.25 | 22.98 | 15.93 | 48.94 |
| MAT(2.5)[S1] | 12.86 | 11.12 | 15.32 | 19.35 | 18.98 | 15.64 | 39.98 |
| MAT(2.5)[S2] | 14.44 | 11.23 | 20.43 | 21.21 | 22.98 | 15.92 | 48.93 |
| MAT(2.5)[S3] | 14.44 | 11.23 | 20.43 | 21.21 | 22.98 | 15.93 | 48.93 |

kernel when we observe the performance on the test set. The surprise detection task had the most uniform performance across all models by this metric. However, the task that was estimated to achieve the best performance is once again the disgust task with the models using GloVe. Anger followed closely with LOO-CV RMSE predictions being just slightly worse for the Glove models. The other tasks, fear, joy, and sadness, were a mixed bag and generally not very good. The valence score LOO-CV RMSE metrics were similar embedding-wise across all kernels. The models using Glove had comparable performance to the STR[T3] models.

Investigation of the test set RMSE results in Table 7.6 show a similar pattern of lighter coloration of the table in every third row. Disgust was apparently the *easiest* task as it had some of the lowest RMSE metrics. The disgust and surprise tasks had the most uniform performance across all models. The anger, disgust, fear, and joy tasks all had uniformly better performance than predicted by the LOO-CV RMSE metrics. Meanwhile, the surprise and valence tasks had uniformly worse performance than predicted by the LOO-CV. It is worth noting the performance of the surprise task for POLY[S1] is poor relative to POLY[S2] and POLY[S3], but this cannot be explained. The valence score test set RMSE metrics notably did not prove to be much better than the test set RMSE for STR[T3].

Examination of the Pearson $r$ coefficients in Table 7.7 strongly suggests that GloVe is a superior embedding choice compared to this implementation of the Doc2Vec embeddings. These metrics further suggest that all kernels with the GloVe embeddings have comparable performance, though the GPs equipped with the Matérn kernels seem to do the best. The tasks that did the best, according to this metric, were fear, valence, and sadness. Disgust surprisingly had some of the lowest $r$ scores despite the low RMSE metrics.

Each kernel with the Doc2Vec embeddings have very similar performance, but all were really bad according to correlation metric. This is an interesting result given that in the disgust task, for example, the test set RMSE metrics are all very comparable to the kernels with the GloVe embeddings. The polynomial kernel predictions are examined in Figure 7.2. When looking at the prediction plots, it is clear that there is detectable correlation in the prediction and true target values for the GloVe models. It also makes sense why the $r$ coefficients are basically zero for the models using Doc2Vec as the predictions and true target values are quite

Table 7.7: Pearson $r$ correlation coefficients between the mean predictions and the test set for the independent single-task GPs. *Light-to-dark* was applied to all the GPs at once. Bigger (darker) scores are better.

| | Anger | Disgust | Fear | Joy | Sadness | Surprise | Valence |
|---|---|---|---|---|---|---|---|
| LIN[S1] | 0.47 | 0.32 | 0.64 | 0.45 | 0.54 | 0.25 | 0.56 |
| LIN[S2] | 0.02 | -0.02 | -0.01 | 0.02 | 0.00 | 0.01 | 0.02 |
| LIN[S3] | 0.03 | -0.02 | 0.00 | 0.02 | 0.00 | 0.01 | 0.02 |
| POLY[S1] | 0.46 | 0.31 | 0.65 | 0.42 | 0.53 | 0.19 | 0.56 |
| POLY[S2] | 0.01 | 0.02 | -0.02 | -0.05 | 0.00 | 0.00 | -0.03 |
| POLY[S3] | 0.00 | 0.02 | -0.02 | -0.05 | -0.01 | 0.00 | -0.03 |
| RBF[S1] | 0.49 | 0.33 | 0.66 | 0.45 | 0.56 | 0.28 | 0.58 |
| RBF[S2] | 0.02 | -0.02 | -0.01 | 0.02 | 0.00 | 0.01 | 0.01 |
| RBF[S3] | 0.03 | -0.02 | 0.00 | 0.02 | 0.00 | 0.01 | 0.02 |
| MAT(0.5)[S1] | 0.51 | 0.34 | 0.67 | 0.47 | 0.57 | 0.29 | 0.59 |
| MAT(0.5)[S2] | 0.02 | -0.01 | -0.02 | -0.03 | -0.01 | 0.01 | -0.01 |
| MAT(0.5)[S3] | 0.02 | -0.01 | -0.02 | -0.02 | -0.01 | 0.00 | 0.00 |
| MAT(1.5)[S1] | 0.50 | 0.34 | 0.67 | 0.46 | 0.57 | 0.29 | 0.58 |
| MAT(1.5)[S2] | 0.02 | -0.01 | -0.01 | 0.00 | 0.00 | 0.01 | 0.01 |
| MAT(1.5)[S3] | 0.03 | -0.01 | -0.01 | 0.00 | 0.00 | 0.01 | 0.01 |
| MAT(2.5)[S1] | 0.50 | 0.33 | 0.66 | 0.45 | 0.56 | 0.29 | 0.58 |
| MAT(2.5)[S2] | 0.03 | -0.01 | -0.01 | 0.01 | 0.00 | 0.01 | 0.01 |
| MAT(2.5)[S3] | 0.03 | -0.01 | -0.01 | 0.01 | 0.00 | 0.01 | 0.02 |

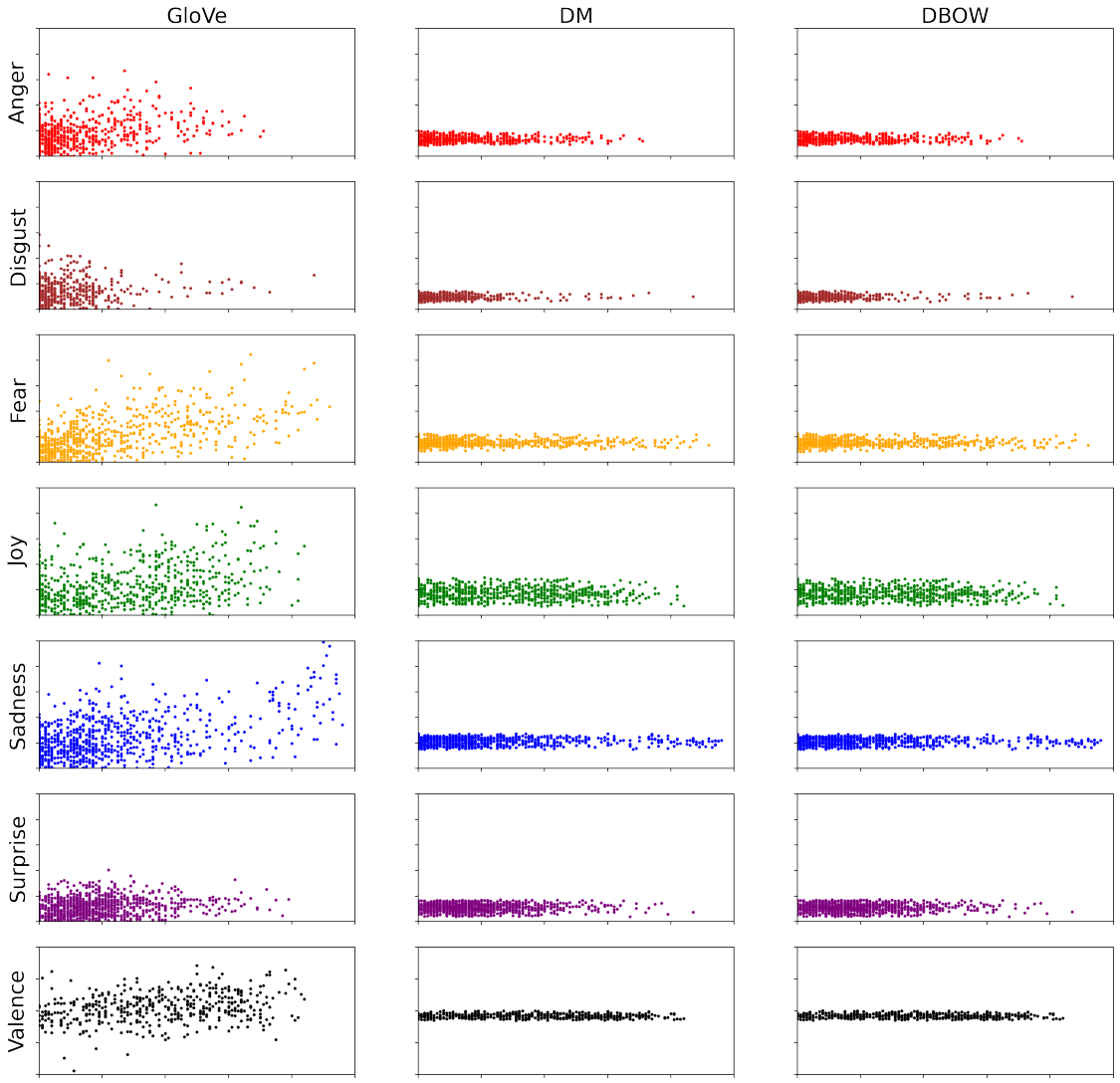


Figure 7.2: Scatter plots of test set predictions for the independent single-task GPs equipped with the polynomial kernels. The horizontal axes are the true target values, and vertical axes are the predicted values. A positive linear trend between the true target values and predictions is desirable.

apparently uncorrelated. This is an interesting insight that the test set RMSE metrics tell one story but the Pearson $r$ correlation coefficients tell another.

Table 7.8 shows some sample predictions made by the independent single-task GP models with MAT(0.5)[S1] and MAT(0.5)[S2]. In these examples, there are several instances of predictions that are pretty close, within a margin of 5 points, and other times where they are way off, by a margin of over 40. We should be cautious to not generalize performance based on a few examples, but these help to illustrate what are models are doing and provide a glimpse at how well they are doing it.

### 7.5.3 Forward Model Selection

We saw that the GPs equipped with the Matérn kernels with varying hyperparameter $\nu$ have roughly comparable performance, but the Matérn kernels with hyperparameter $\nu = 0.5$ tended to do the best out of the three overall. For this reason, we are only going to consider the kernel MAT(0.5) and are going to drop the hyperparameter $\nu$ from our notation. Thus, from this point on, we will now only consider the following subset of kernels for the rest of our analysis:

$$\begin{aligned} \{ &\text{LIN[S1]}, \quad \text{LIN[S2]}, \quad \text{LIN[S3]}, \\ &\text{POLY[S1]}, \quad \text{POLY[S2]}, \quad \text{POLY[S3]}, \\ &\text{RBF[S1]}, \quad \text{RBF[S2]}, \quad \text{RBF[S3]}, \\ &\text{MAT[S1]}, \quad \text{MAT[S2]}, \quad \text{MAT[S3]} \}. \end{aligned} \tag{7.12}$$

For this procedure, we will use forward stepwise selection to produce new compound kernels (by means of kernel addition or multiplication) to see if we can improve performance. We will proceed to conduct forward selection of GPs on a task-by-task basis and use the LOO-CV RMSE as the error metric of choice in our model

Table 7.8: Random sample of test predictions made by the independent single-task GPs equipped with Matérn kernels ($\nu = 0.5$). The values in the first, second, and third rows are the true target values, predictions for the GPs with MAT(0.5)[S1], and predictions for the GPs with MAT(0.5)[S2], respectively.

| Headline | | Ang | Dis | Fear | Joy | Sad | Sur | Val |
|---|---|---|---|---|---|---|---|---|
| Bush vows cooperation | $y_*$ | 0 | 7 | 10 | 60 | 10 | 0 | 71 |
| on health care | [S1]: $\hat{y}_*$ | 4.7 | 4.1 | 9.2 | 26.7 | 15.7 | 9.8 | 9.8 |
| | [S2]: $\hat{y}_*$ | 13.4 | 9.3 | 16.1 | 19.3 | 18.8 | 11.2 | -8.6 |
| Schuey sees Ferrari | $y_*$ | 0 | 0 | 0 | 46 | 0 | 31 | 40 |
| unveil new car | [S1]: $\hat{y}_*$ | 2.5 | 4.3 | 3.4 | 22.7 | 20.2 | 11.1 | 6.4 |
| | [S2]: $\hat{y}_*$ | 12.3 | 8.1 | 15.0 | 22.0 | 17.2 | 9.4 | -6.8 |
| Portugal to vote on | $y_*$ | 2 | 8 | 10 | 20 | 13 | 24 | 12 |
| abortion laws | [S1]: $\hat{y}_*$ | 8.0 | 3.7 | 14.3 | 10.7 | 8.5 | 6.1 | -3.8 |
| | [S2]: $\hat{y}_*$ | 15.0 | 9.8 | 16.9 | 16.1 | 20.5 | 10.3 | -12.0 |
| Evacuees can return | $y_*$ | 6 | 0 | 14 | 51 | 16 | 15 | 54 |
| home after plant blast | [S1]: $\hat{y}_*$ | 13.7 | 3.5 | 35.8 | 14.0 | 41.7 | 11.5 | -31.9 |
| | [S2]: $\hat{y}_*$ | 14.0 | 9.6 | 17.2 | 16.5 | 20.1 | 11.0 | -10.1 |
| Congressman rebukes | $y_*$ | 61 | 15 | 4 | 0 | 23 | 0 | -69 |
| U.S. allies for lack of | [S1]: $\hat{y}_*$ | 23.7 | 15.1 | 24.7 | 15.3 | 17.4 | 8.5 | -17.4 |
| support in Afghanistan | [S2]: $\hat{y}_*$ | 13.4 | 10.1 | 16.3 | 19.6 | 22.9 | 12.5 | -9.7 |

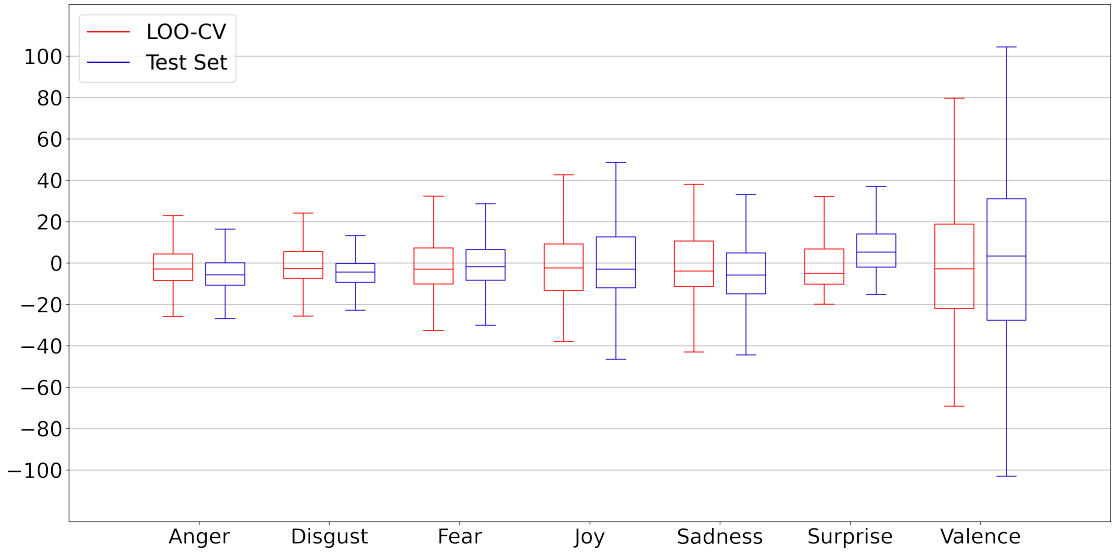


Figure 7.3: Residual boxplots for independent GPs equipped with the Matérn kernel using the GloVe embeddings.

selection. Moreover, we will iteratively compound kernels until there is no further improvement by more than 3%. In this forward selection, we will skip the following compounds to avoid overparameterization:

$$\text{LIN} + \text{LIN}, \text{RBF} \times \text{RBF}, \text{ and } \text{MAT} \times \text{MAT}.$$

### Results

It turns out that this stop condition resulted in the MAT[S1] kernel being selected in all cases. That is, when assessing the independent GPs for each task, the performance of the MAT[S1] was not surpassed by more than 3% in the first iteration of forward model selection. This is strong evidence that a GP equipped with MAT[S1] is not only sufficient, but proficient when it comes to building a model for conducting emotion and sentiment analyses in our context. This warrants doing some extra analysis regarding this kernel.

The box plots in Figure 7.3 are approximately symmetric for the LOO-CV and test set residuals. Related to predictions on the test set, it appears that the models tend to overestimate anger, disgust, and sadness scores, while they tend to underestimate surprise scores. Moreover, the GPs predicting on test set scores for fear, joy, and valence have median residuals close to 0. The residuals for the test prediction concerning joy in particular has the greatest spread of all the emotions and the most skewed residual box plot with the third interquartile range being larger than the second. The test set residuals for the valence score have a significantly large spread with errors surpassing 100 in both directions, however, 50% of the data is predicted decently well with a margin of error of about 30.

### 7.5.4 Compound Kernels

We are going to repeat our forward model selection procedure, but this time we are going to relax our stop condition so that there is a chance to build some interesting compound kernels. This time, we will continue to compound kernels until there is no further improvement in the LOO-CV RMSE. In relaxing this condition, this will allow us to see how a new model can be created that is technically better according to the conditions set in place but also demonstrate how this can make the model prone to overfitting.

#### Results

The results for this procedure can be found in Table 7.9. Compounding kernels with forward selection should have theoretically produced better models because each of these had lower LOO-CV RMSE metrics. However, the importance of a good stop condition is illustrated in this example. The LOO-CV RMSE metrics are only

Table 7.9: RMSE metrics produced after fitting the independent single-task GPs equipped with compound kernels obtained through forward selection while using a weak stop condition. The kernels are presented in list processing (LISP) symbolic-expressions (S-expressions) so that the order in which the kernels were compounded can be recovered by reading it from right to left. Note that GloVe is used in all cases other than the cases where specified.

| Task | Kernel S-expression | LOO-CV | Test Set |
|---|---|---|---|
| Anger | +(MAT,+(MAT,+(MAT,MAT))) | 14.25 | 12.73 |
| Disgust | MAT | 13.17 | 11.06 |
| Fear | +(MAT,+(RBF,MAT)) | 18.23 | 15.24 |
| Joy | +(MAT,+(MAT,+(POLY[S2],MAT))) | 19.83 | 19.06 |
| Sadness | +(MAT,+(MAT,+(LIN[S2],MAT))) | 18.30 | 18.94 |
| Surprise | ×(RBF,MAT) | 14.47 | 15.86 |
| Valence | +(POLY[S2],MAT) | 30.84 | 40.07 |

marginally better than those for the MAT[S1] kernel. More importantly, the test set RMSE metrics for these models are all higher than the test set RMSE metrics for models with MAT[S1]. This example shows that the LOO-CV is subject to fallibility in terms of estimating the test set RMSE and, more importantly, that a weak stop condition in forward selection can be counter-productive. Notice that most of the models include additional MAT[S1] components in their compound kernels. This suggests that the key component in these kernels is indeed the MAT[S1] kernel. Ultimately, the MAT[S1] kernel has better performance than the compound kernels across all tasks and is sufficient on its own.

## 7.6 Multi-task GPs

In this section we will employ multi-task GPs to learn and predict for our tasks. There is the potential to achieve relatively superior models given that they can learn from correlations between tasks. This will also benefit us in that we will be able to inspect the covariance matrices of inter-task relationships. For the multi-task GPs, we will now have to consider the ranks of the kernels, adding an additional level of complexity. Once again, let us fix $\tau = 100$.

### 7.6.1 Basic Kernels

First, we will start by using basic kernels to see if there is any substantial improvement when incorporating intertask learning in our GPs. We will cycle through the refined set of kernels in the set 7.12 as well cycle through the ranks ranging from 2 to 7 inclusive. This will result in us examining a total of $12 \times 6 = 72$ different model constructions.

**Results**

The results for these models are displayed in Figure 7.4. The results serve as more evidence that GPs equipped with MAT[S1]-$\rho$ for all ranks $2 \leq \rho \leq 7$ are superior models. Not only is the LOO-CV RMSE generally lower than all the other kernels for all ranks, but it achieved the best test set RMSE metrics too. On the basis of kernels, the polynomial kernel with the Doc2Vec embeddings actually outperforms the polynomial kernel with the GloVe embedding. This is interesting because the LOO-CV for POLY[S1] suggests that it would do better than the others, but it in fact does worse. The linear and RBF kernels show the same behavior that the ones with GloVe embeddings do better by LOO-CV and test set RMSE metrics.

When it comes to the performance of the multi-task GPs with the Matérn kernels ($\nu = 0.5$), the performance of these models fare very closely and is comparable to the single-task GPs. There is one notable difference, however. The surprise task actually slightly fares worse in the multi-task case which is supported by the fact that the Pearson $r$ values are little smaller than for the single-task case. This suggests that using the independent single-task GPs for the surprise task in particular may be the preferable option.

Given that we are doing multi-task learning, we also now have the option to inspect the inter-task similarities. Examination of the inter-task correlation matrix for the GP equipped with MAT[S1]-7 in Figure 7.5 shows a very close resemblance to the correlation matrix of the data shown in Figure 7.1 meaning that the GP was able to pick up on the correlations in the data. The tasks sharing the highest degrees of correlation with valence were joy and sadness, as speculated. Anger and disgust share a very high correlation, which is rather unsurprising. Anger, disgust, and fear show negative correlation with valence, as was to be expected, but these

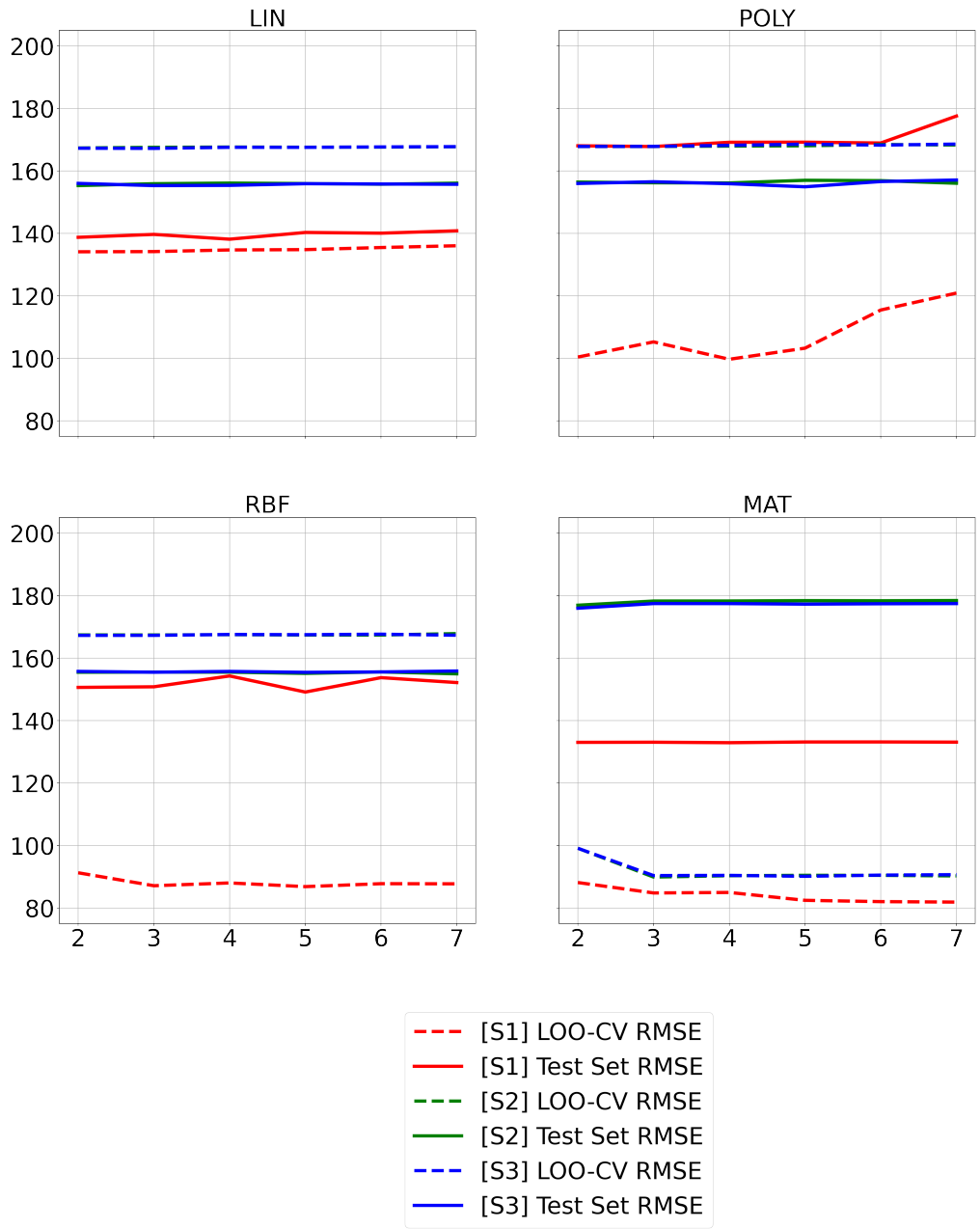


Figure 7.4: Plots of aggregate RMSE metrics for various kernels and ranks. The horizontal axes are the ranks of the kernels, and the vertical axes are the measures for the aggregate RMSE metrics.


Figure 7.5: Inter-task correlation matrix produced by the GP equipped with MAT[S1]-7. *Light-to-dark* was applied to values in this matrix. Greater (darker) values indicates more correlation between values, and lesser (lighter) values indicate more anti-correlation between values.

Table 7.10: Random sample of test predictions made by the multi-task GP equipped with MAT[S1]-7. The first and second rows are the true target values and the predictions for the GP, respectively.

| Headline | | Ang | Dis | Fear | Joy | Sad | Sur | Val |
|---|---|---|---|---|---|---|---|---|
| London is pace-setter on | $y_*$ | 5 | 3 | 7 | 27 | 10 | 6 | 16 |
| congestion toll, says mayor | $\hat{y}_*$ | 17.8 | 13.0 | 19.6 | 1.7 | 37.6 | 3.1 | -39.5 |
| High cholesterol raises | $y_*$ | 0 | 0 | 45 | 0 | 17 | 23 | -46 |
| stroke risk healthy women | $\hat{y}_*$ | 3.2 | 1.8 | 16.1 | 23.5 | 17.7 | 12.0 | -0.4 |
| Senate votes to | $y_*$ | 13 | 0 | 18 | 0 | 35 | 14 | -52 |
| revoke pensions | $\hat{y}_*$ | 4.2 | 1.8 | 8.4 | 8.1 | 8.5 | 4.4 | -4.7 |
| Women protest | $y_*$ | 36 | 26 | 13 | 0 | 26 | 31 | -14 |
| Pakistan demolition | $\hat{y}_*$ | 32.4 | 22.2 | 38.2 | 19.6 | 40.5 | 0.7 | -34.8 |
| Pig cells hope for | $y_*$ | 0 | 0 | 3 | 35 | 2 | 38 | 54 |
| diabetes cure | $\hat{y}_*$ | 0.9 | -1.0 | 22.6 | 20.6 | 14.3 | 21.2 | -1.6 |
| For women, aspirin a day | $y_*$ | 0 | 0 | 7 | 41 | 6 | 21 | 47 |
| could keep stroke away | $\hat{y}_*$ | 4.5 | 4.4 | 11.0 | 25.7 | 18.8 | 12.7 | 1.6 |
| House ethics panel talks | $y_*$ | 0 | 0 | 0 | 29 | 0 | 12 | 34 |
| to Hastert aide | $\hat{y}_*$ | 4.2 | 5.3 | -2.9 | 8.2 | 6.3 | 4.8 | 2.2 |
| Iran says Sunnis, using | $y_*$ | 19 | 9 | 83 | 0 | 16 | 16 | -86 |
| Pakistan as base, planned | $\hat{y}_*$ | 35.3 | 25.8 | 46.4 | 0.6 | 44.8 | 6.2 | -55.6 |
| fatal bombing | | | | | | | | |

are not as strong. It comes as a surprise how much the surprise task is correlated with valence and joy. With that, it is difficult to reason why it does not have a stronger anti-correlation with sadness.

### 7.6.2 LMC with Matérn Kernel

For this analysis, we will implement GP models using the MAT[S1] kernel obtained under the LMC assumption. In this context, we now need to consider the noise rank of the matrix in addition to the kernel rank. We will proceed to iterate over the kernel ranks from 1 to 7 inclusive and noise ranks from 0 to 7 inclusive. We fix $\tau = 200$ for increased opportunity for hyperparameter optimization. Once the models are fit, we will then produce aggregate RMSE metrics and see how our models do.

#### Results

The results for this analysis are found in Figure 7.6. The LOO-CV metrics suggest that increasing the noise rank does not help the predictive performance of the models. It is rather surprising to see that in the noise rank-1 cases there are several models that perform relatively poorly according to this metric. According to this table, the aggregate LOO-CV RMSE metrics suggest arguing 0 for the noise rank.

The test set RMSE metrics say something different. It is the case that all the scores are very comparable. This may suggest that the noise is not very strong and thus does not influence prediction all that much. Another possibility is that there is not a lot of structure within the data for the models to pick up on.

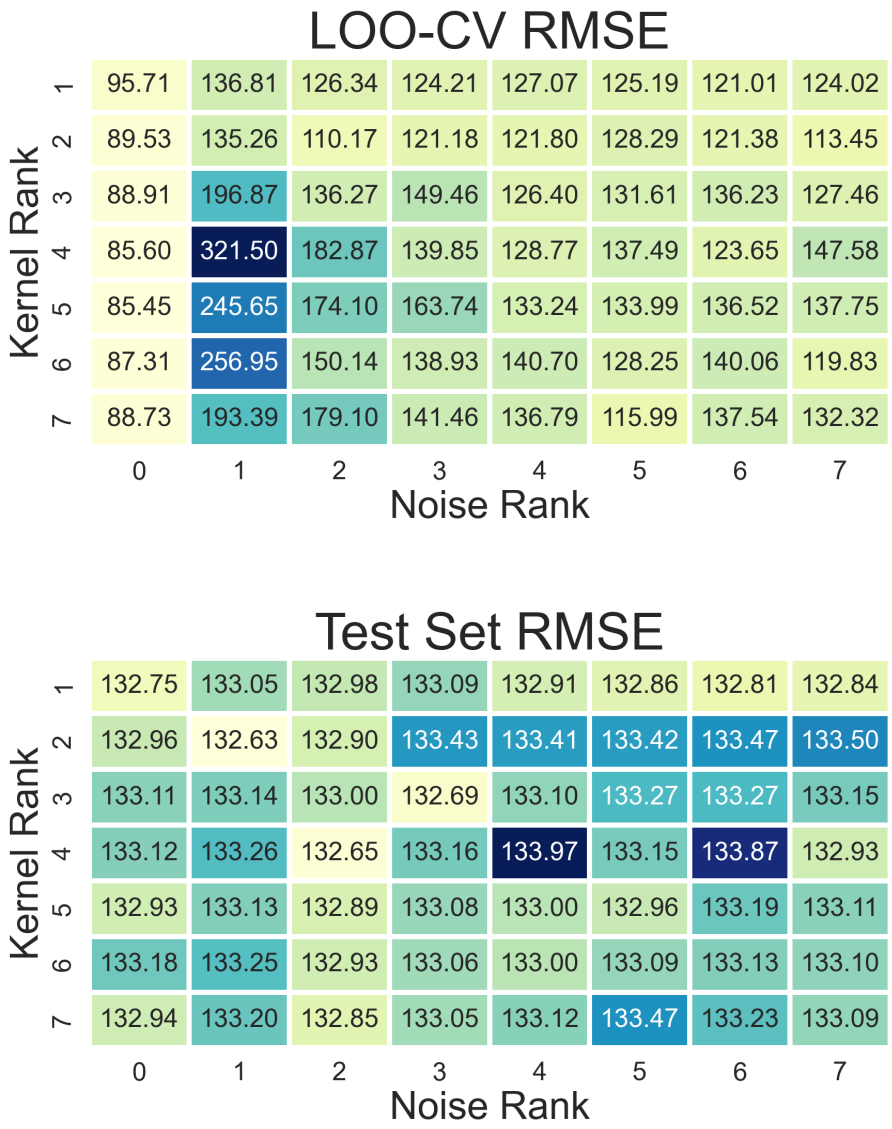


Figure 7.6: LOO-CV and test set RMSE metrics for the multi-task GP equipped with the LMC MAT[S1] kernel with the different task ranks and noise ranks. The horizontal axis is the noise ranks and vertical axes are the task ranks. *Light-to-dark* was applied to all the GPs at once but to each set of metrics separately. Smaller (lighter) scores are better.

### 7.6.3 LMC with Forward Selection

For this procedure, we will use forward model selection to build an LMC kernel composed of basic kernels and compounded kernels to iteratively build more complicated models until a superior model is achieved. We have empirical evidence that the MAT[S1] kernel does well and will serve as a good starting point in the model selection process, so we begin with MAT[S1]-1 with noise rank-0.

Our forward model selection process will entail (1) iterating over all of the basic kernels in the set 7.12 and the compound kernels produced in Table 7.9 to include in the LMC kernel, (2) incrementing the kernel ranks over the existing kernels already present in the LMC, where possible, and (3) incrementing the noise rank of the LMC. It is important to recognize that the rank of any component kernel in the LMC cannot exceed 7. If it is the case that all of the component kernels (possibly only one) reaches rank 7 before a new kernel has been added, then the next iteration will try adding all of the basic and compound kernels, try incrementing the noise rank, and go without incrementing the ranks of any of the component kernels. Moreover, an additional constraint is that the noise cannot exceed 7. We disallow this rank from incrementing if the max rank is reached. For this procedure, we set $\tau = 100$. Once again, the aggregate LOO-CV RMSE metrics will be compared during each iteration. This procedure will continue until there is no further improvement in this error metric.

#### Results

The result of this forward model selection is a noise rank-0 LMC kernel with only MAT[S1]-4 as its component kernel, where rank-0 noise in this case means that the noises are independent. This result supports the claim that the Matérn kernel

Table 7.11: Error metrics for the GP with MAT(0.5)[S1]-4 and independent noises.

MAT(0.5)[S1]-4 with independent noises

| Ang | Dis | Fear | Joy | Sad | Sur | Val |
|---|---|---|---|---|---|---|

LOO-CV RMSE

| | | | | | | |
|---|---|---|---|---|---|---|
| 12.21 | 10.68 | 11.05 | 11.98 | 12.89 | 13.85 | 11.93 |

Test Set RMSE

| | | | | | | |
|---|---|---|---|---|---|---|
| 13.10 | 11.44 | 15.37 | 19.20 | 19.20 | 15.81 | 39.87 |

Test Set Pearson $r$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.49 | 0.33 | 0.66 | 0.46 | 0.56 | 0.23 | 0.58 |

($\nu = 0.5$) with the GloVe embedding is robust and a consistent model for the emotion and sentiment analyses tasks.

The resulting error metrics for this model are given in Table 7.11. As it happens, the LOO-CV RMSE metrics for this model are quite low. And thus, the expectation is that the model will do well, come time of prediction. However, it seems that the model has been overfitted as this model does the same or worse than its independent single-task counterparts as can be seen by comparing these results to the results in Figures 7.6 and 7.7.

# Chapter 8

# Conclusion and Final Remarks

To conclude this thesis, we will discuss the results of our analysis and things we can take away from it. Then, we will talk about possible avenues of future research and applications of the research presented in this thesis.

## 8.1 Results and Discussion

The overall results of our analysis demonstrate that it is possible to predict scores of affect based on news headlines. How well this is done depends on the embedding method employed and the kernel function being used.

One result that we found is that is is possible to detect emotions with elementary hard matching approaches. This is best showcased by the GPs equipped with the string kernels using the one-hot encoding schemes. Despite the relative simplicity of these hard matching embeddings, they were able to achieve non-trivial results with respect to the RMSE metrics and even the Pearson $r$ correlation coefficients. The takeaway is that the one-hot encodings have the capacity to work in conjunction with the string kernel. The fact that these models had decent results in our setting

with limited contextual information is promising and begs further investigation.

The string kernel had much better performance when using the Glove embeddings and was even able to produce some rather competitive results despite the relatively basic instantiations of the models. This can be verified by examining the independent single-task models with STR[T3] and MAT(0.5)[S1] in Table 8.1. It is unfortunate that not many models could incorporate this kernel due to computational limitations for it would have been interesting to test different hyperparameters, include it in compound kernels, and use it in multi-task models. All said and done, the string kernel did pretty well and warrants future investigation in NLP.

Our data analysis with the sequential token embeddings provided us with some more competitive models. In the single-task case, the models we studied showed that the Matérn kernel with $\nu = 0.5$ using the GloVe embeddings was a top contender, where the other Matérn kernels with $\nu = 1.5$ and $\nu = 2.5$ showed similar performance. In fact, all other single-task models equipped with the linear, polynomial, and RBF kernels using the GloVe embeddings had pretty comparable performance to MAT(0.5)[S1].

The Doc2Vec embeddings did not prove to be as proficient as the GloVe embeddings in this setting. Our results in the single-task case suggests that most of their guesses were lucky and were not truly learning much from the model. This can be extrapolated by examining the $R^2$ values obtained by squaring the Pearson $r$ correlation coefficients in Table 7.7. All the models using Doc2Vec had $R^2$ values that were basically zero, meaning that no variability in the target data could be explained by the models. Given that the GloVe embeddings were pre-trained on a corpus prior to their use, it would be unfair to compare the performance of two embeddings in this context with such limited training data to determine which is

Table 8.1: Results for the independent single-task GP models equipped with STR[T3] and MAT(0.5)[S1].

| Kernel | Ang | Dis | Fear | Joy | Sad | Sur | Val |
|---|---|---|---|---|---|---|---|
| LOO-CV RMSE | | | | | | | |
| STR[T3] | 15.79 | 14.46 | 18.55 | 22.79 | 20.37 | 15.07 | 33.18 |
| MAT(0.5)[S1] | **14.25** | **13.18** | **18.23** | **19.88** | **18.31** | **14.66** | **30.96** |
| Test Set RMSE | | | | | | | |
| STR[T3] | 12.73 | **10.89** | 15.95 | 20.67 | 19.78 | 16.18 | 40.28 |
| MAT(0.5)[S1] | **12.68** | 11.02 | **15.20** | **18.89** | **18.87** | **15.67** | **39.88** |
| Test Set Pearson $r$ | | | | | | | |
| STR[T3] | 0.49 | **0.34** | 0.64 | 0.36 | 0.52 | 0.28 | 0.57 |
| MAT(0.5)[S1] | **0.51** | **0.34** | **0.67** | **0.47** | **0.57** | **0.29** | **0.59** |

the better embedding algorithm in general. It is fair to say, however, that this setting in which we implemented Doc2Vec simply did not have enough information for the embedding algorithm to learn from.

We implemented compound kernels to try to achieve better models. Unfortunately, it did not prove to be very fruitful as the test set RMSE metrics were all higher than the single-task MAT(0.5)[S1] models. In hindsight, the LOO-CV actually played a role in overfitting models because of the metrics that it produced. If we were in a situation where the true target outputs of the test set were unknown, then we would not have any reason not to trust the LOO-CV RMSE metrics to guide our model selection.

Our multi-task models demonstrated that the models with MAT(0.5)[S1]-$\rho$ were superior to the other models we examined. The multi-task learning method proved to be insightful in what the model learns about the similarities between tasks. Based on our inter-task correlation matrix produced by the GP with MAT(0.5)[S1]-7 in Figure 7.5, it was able to closely recover the correlation matrix of the training data in Figure 7.1. The sample of test set predictions in Table 7.10 demonstrate that the predictions were not trivial but not great either. Some predicted scores are pretty close, while others really miss the mark.

The implementation of our LMC models showed that introducing distinct latent functions and noises matrix did not really help with producing a better model by test set RMSE metrics. This kind of model was most likely too advanced for the little amount of data we were working with and needed more to really be effective. Even in this most complicated setting, the model that we end up with is a variation of the MAT(0.5)[S1] kernel.

One insight that can be extracted from our analysis is the importance of embed-

dings. It seems that a lot of predictive power comes from the embedding approach that is used when it comes to employing kernel methods for emotion and sentiment analyses. It seems fair to conjecture that repeating this analysis with other pre-trained embeddings such as Word2Vec or Doc2Vec would result in new and interesting models for giving predictions.

## 8.2 Reflections

The use of multi-task kernels in this setting did not yield improved performance compared to the single-task GPs. This can most likely be attributed to the lack of data for the GPs to learn from. Thus, the analysis performed in this thesis would benefit from more data for training. Had there been a larger training set, it is almost guaranteed that we would have seen much more interesting results from higher predictive accuracy for the Matérn kernels, to improved performance with the Doc2Vec embeddings, to more unique multi-task models. It is also unclear if the training set was representative of the test set, which could have biased the GPs during training. The reasoning for this claim is because of the numerous 0-valued emotion scores in the development set and the significant amount of *weak* scores.

A statement should be made about the subjectivity of the tasks at hand. Emotions are a tricky subject, and not everyone feels things the same way due to the idiosyncratic nature of individuals. It is almost certain that the scores provided in the test dataset were not unanimous and were averaged in some manner. This means that affective states caused by a stimulus follow a distribution. It is reasonable to conjecture that this distribution is normal, so GPs may indeed be the best way to try to capture that. All said and done, given how little data we had to train

with, our models did okay in predicting scores of affect. Some amount of error is to be expected given that GPs were meant to model emotion and valence, phenomena that are inherently subjective.

## 8.3 Future Work

More work can certainly be done with the string kernel. The string kernel is a complex but potentially powerful tool to use in NLP. Despite the limited use of the kernel in our analysis, its performance was on par with the other kernels. It is a very interesting and innovative kernel computation that is deserving of more attention. One procedure that would be interesting to do in the future is to employ random search over the hyperparameter values. This could potentially lead to better model performance and would bypass the training stage of model development. GPyTorch [24] has the functionality for the user to integrate PyTorch tensors stored in CUDA memory, which allows for use of a local GPU device to perform tensor computations [46]. Implementing the string kernel while using CUDA in conjunction with GPyTorch's Lazy tensors is likely to improve the speed of kernel computations and thus the ability to conduct more experiments. When computational complexity is less of an issue, one avenue of exploration will be to extend the number of $n$-gram coefficients that the kernel uses. Moreover, the kernel could use symbol-dependent decay factors containing information from outside datasets to likely give even better performance. This would be an interesting kernel to use in compound kernels and in LMC kernels. The biggest hurdle is the time complexity. So if this can be overcome, then there is so much room for exploration.

It is most likely the case that the models using Doc2Vec would have done better

if there was more information for it to learn from. Seeing how these representations of the headlines were rather lacking in information, it would be interesting to see how well GP models would do if they used Doc2Vec embeddings trained on the whole news articles that would be representative of the headlines. This would require cleaning each article to ensure that they are text processable, and the embeddings would not benefit from images that may be important to the news pieces. Nevertheless, representing the news headlines with vectors holding information about the article passages could potentially lead to massive boosts in performance. Although this kind of exploration could prove to be fruitful, it is not directly comparable to the tasks attempted in this thesis. The reason for this is that information was meant to be extracted solely from the headlines, which are characteristically short and lacking in information. The incorporation of additional information from the articles is a way to supplement this but defies the constraints of the task done here.

The framework of this analysis could be applied in studies of other embedding algorithms and kernels. Ideally, a new and larger dataset would be available to work with, but the concept still holds. The methods employed here to combine and mix kernels were done with a few kernels where we had a couple of shiners. It would be quite interesting to see how this would work with other state-of-the-art embeddings and other kernel functions.

An exciting aspect of researching this task in NLP is that there is a lot of room for working on material that is not related to news headlines. The sentiment analysis task performed here could easily extend to other applications such as opinion mining for online reviews. This is a major application in today's market with our current and growing reliance on the internet and e-commerce. The emo-

tion analysis task is also in its primitive stages as there are many other expressive emotional families not even mentioned in this dataset [19]. One aspect that would be particularly interesting is to study the inter-task correlation matrices between complex emotional states. This could provide empirical insights that lead to theoretical connections between affective states and provide a basis for computational psychology.

# Bibliography

[1] Asma Ben Abacha and Pierre Zweigenbaum. Means: A medical question-answering system combining nlp techniques and semantic web technologies. *Information processing & management*, 51(5):570–594, 2015.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[3] Edgar Altszyler, Mariano Sigman, Sidarta Ribeiro, and Diego Fernández Slezak. Comparative study of lsa vs word2vec embeddings in small corpora: a case study in dreams database. *arXiv preprint arXiv:1610.01520*, 2016.

[4] Mauricio A Alvarez, Lorenzo Rosasco, and Neil D Lawrence. Kernels for vector-valued functions: a review. *stat*, 1050:16, 2012.

[5] Lisa Feldman Barrett. Discrete emotions or dimensions? the role of valence focus and arousal focus. *Cognition & Emotion*, 12(4):579–599, 1998.

[6] Daniel Beck. Gaussian processes for text regression. 2017.

[7] Jerome Bellegarda. Emotion analysis using latent affective folding and embedding. In *Proceedings of the NAACL HLT 2010 workshop on computational approaches to analysis and generation of emotion in text*, pages 1–9, 2010.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[9] Edwin V Bonilla, Kian Chai, and Christopher Williams. Multi-task gaussian process prediction. *Advances in neural information processing systems*, 20, 2007.

[10] Sven Buechel and Udo Hahn. Emobank: Studying the impact of annotation perspective and representation format on dimensional emotion analysis. *arXiv preprint arXiv:2205.01996*, 2022.

[11] Nicola Cancedda, Eric Gaussier, Cyril Goutte, and Jean Michel Renders. Word sequence kernels. *The Journal of Machine Learning Research*, 3:1059–1082, 2003.

[12] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[13] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. A survey on dialogue systems: Recent advances and new frontiers. *Acm Sigkdd Explorations Newsletter*, 19(2):25–35, 2017.

[14] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[15] Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[16] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[17] Morris L.. Eaton. *Multivariate Statistics. A Vector Space Approach.-A Volume in the Wiley Series in Probability and Mathematical Statistics*. Wiley-Interscience, 1983.

[18] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4): 169–200, 1992.

[19] Paul Ekman. Basic emotions. *Handbook of cognition and emotion*, 98(45-60): 16, 1999.

[20] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[21] Nico H Frijda et al. *The emotions*. Cambridge University Press, 1986.

[22] Mahak Gambhir and Vishal Gupta. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, 47(1):1–66, 2017.

[23] Alfio Gliozzo and Carlo Strapparava. *Semantic domains in computational linguistics*. Springer Science & Business Media, 2009.

[24] GPyTorch. Documentation: Gpytorch. URL `https://docs.gpytorch.ai/en/stable/`.

[25] Nida Manzoor Hakak, Mohsin Mohd, Mahira Kirmani, and Mudasir Mohd. Emotion analysis: A survey. In *2017 international conference on computer, communications and electronics (COMPTELIX)*, pages 397–402. IEEE, 2017.

[26] David Harris and Sarah L Harris. *Digital Design and Computer Architecture*. Elsevier, 2012.

[27] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[28] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[29] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[30] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[31] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[32] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA., 2009.

[33] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume Third Edition draft. Prentice Hall, 2020.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Mei Kobayashi and Koichi Takeda. Information retrieval on the web. *ACM Computing Surveys (CSUR)*, 32(2):144–173, 2000.

[36] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*, 2016.

[37] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[38] Elizabeth D Liddy. Natural language processing. 2001.

[39] Huma Lodhi, John Shawe-Taylor, Nello Cristianini, and Christopher Watkins. Text classification using string kernels. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL `https://proceedings.neurips.cc/paper/2000/file/68c694de94e6c110f42e587e8e48d852-Paper.pdf`.

[40] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of machine learning research*, 2(Feb):419–444, 2002.

[41] Kaustubh Mani, Ishan Verma, Hardik Meisheri, and Lipika Dey. Multi-document summarization using distributed bag-of-words model. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 672–675. IEEE, 2018.

[42] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL `https://arxiv.org/abs/1301.3781`.

[43] Mandar Mitra and BB Chaudhuri. Information retrieval from documents: A survey. *Information retrieval*, 2(2):141–163, 2000.

[44] Tetsuya Nasukawa and Jeonghee Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77, 2003.

[45] NLTK. Documentation: Nltk package. URL `https://www.nltk.org/api/nltk.html`.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran As-

sociates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[47] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[48] Marie-Therese Puth, Markus Neuhäuser, and Graeme D Ruxton. Effective use of pearson's product–moment correlation coefficient. *Animal behaviour*, 93: 183–189, 2014.

[49] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[50] J.O. Rawlings, S.G. Pantula, and D.A. Dickey. *Applied Regression Analysis: A Research Tool*. Springer Texts in Statistics. Springer New York, 2006. ISBN 9780387227535. URL https://books.google.se/books?id=M1MFCAAAQBAJ.

[51] Seyed Mahdi Rezaeinia, Rouhollah Rahmani, Ali Ghodsi, and Hadi Veisi. Sentiment analysis based on improved pre-trained word embeddings. *Expert Systems with Applications*, 117:139–147, 2019.

[52] Ronald Rivest. The md5 message-digest algorithm. Technical report, 1992.

[53] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[54] Magnus Sahlgren. The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53, 2008.

[55] Harold Schlosberg. Three dimensions of emotion. *Psychological review*, 61(2): 81, 1954.

[56] SciKit-Learn. Documentation: Scikit-learn preprocessing onehotencoder. URL `https://https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html`.

[57] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[58] Jacopo Staiano and Marco Guerini. Depechemood: a lexicon for emotion analysis from crowd-annotated news. *arXiv preprint arXiv:1405.1605*, 2014.

[59] Carlo Strapparava and Rada Mihalcea. SemEval-2007 task 14: Affective text. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 70–74, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `https://aclanthology.org/S07-1013`.

[60] Carlo Strapparava, Alessandro Valitutti, et al. Wordnet affect: an affective extension of wordnet. In *Lrec*, volume 4, page 40. Lisbon, Portugal, 2004.

[61] TensorFlow Keras Team. keras/keras/preprocessing/text.py, 2015. URL `https://github.com/keras-team/keras/blob/v2.9.0/keras/preprocessing/text.py#L184-L543`.

[62] Yee Whye Teh, Matthias Seeger, and Michael I Jordan. Semiparametric la-

tent factor models. In *International Workshop on Artificial Intelligence and Statistics*, pages 333–340. PMLR, 2005.

[63] Atousa Torabi, Niket Tandon, and Leonid Sigal. Learning language-visual embedding for movie understanding with natural-language. *arXiv preprint arXiv:1609.08124*, 2016.

[64] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings. *arXiv preprint arXiv:1909.07940*, 2019.

[65] Yanshan Wang, Sijia Liu, Naveed Afzal, Majid Rastegar-Mojarad, Liwei Wang, Feichen Shen, Paul Kingsbury, and Hongfang Liu. A comparison of word embeddings for the biomedical natural language processing. *Journal of biomedical informatics*, 87:12–20, 2018.

[66] Stefan Wermter, Ellen Rilo, and Gabriele Scheler. Connectionist, statistical and symbolic approaches to learning for natural language processing. 1996.

[67] Rui Xia and Zixiang Ding. Emotion-cause pair extraction: A new task to emotion analysis in texts. *arXiv preprint arXiv:1906.01267*, 2019.

[68] Jing Xiao, Tat-Seng Chua, and Hang Cui. Cascading use of soft and hard matching pattern rules for weakly supervised information extraction. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 542–548, 2004.

[69] Ali Yadollahi, Ameneh Gholipour Shahraki, and Osmar R Zaiane. Current state of text sentiment analysis from opinion to emotion mining. *ACM Computing Surveys (CSUR)*, 50(2):1–33, 2017.

# Appendix A

# Conditional Distribution of Multivariate Normal Variables

**Theorem A.0.1.** Eaton [17]

Let $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ such that

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}, \ \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \text{ and } \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}. \tag{A.1}$$

Then, the conditional distribution of $\mathbf{y}_1$ given $\mathbf{y}_2$ is also multivariate normal with

$$\mathbf{y}_1 | \mathbf{y}_2 \sim \mathcal{N}(\boldsymbol{\mu}_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{y}_2 - \boldsymbol{\mu}_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}). \tag{A.2}$$

# Appendix B

# Inverse Document Frequency

The bag of words model can be used to obtain the *inverse documents frequency* (*IDF*) of terms. The IDF is defined as [11]

$$\text{idf}_a = \log\left(\frac{n}{n_a}\right), \tag{B.1}$$

where $n$ is the number of documents in a collection and $n_a$ is the number of documents containing term $a$. $\text{idf}_a$ is bounded between 0 and $\log(n)$ and serves as a way to provide distributional information about a term. When used as a decay factor $\lambda_a$ in the word sequence kernel, the decay factor needs to be normalized:

$$\lambda_a = \frac{\log\left(\frac{n}{n_a}\right)}{\log(n)}. \tag{B.2}$$

# Appendix C

# Preliminary Trial

## C.1 ModelList Multi-Output GP

The ModelList multi-output GP has the advantage that it trains quickly because it optimizes across all tasks at once. Running this model for various numbers of training iterations helped to give us a baseline for the number of training iterations that are necessary to optimize the hyperparameters of a GP.

### C.1.1 Training Iterations

For this trial, we trained the GP for $\tau = 500$ training iterations to observe the error loss of the loss function after each training iteration. This was done for all kernels except the string kernels due to computational limitations of computer hardware.

The purpose of this experiment is to determine the number of training iterations that are appropriate for more computationally expensive experiments. This can be reasonably estimated by observing the rates of convergence for the loss function and LOO-CV RMSE metrics versus training iterations.

**Results**

The plots found in Figure C.1.1 clearly converge in the error loss where it is reached before $\tau = 100$. These results suggest that performing 100 iterations is an adequate number of iterations to fix when it comes to training the GP models. The results are the same for the LOO-CV RMSE metrics such that they all appear to reach relatively stable values by the 100th training iteration.

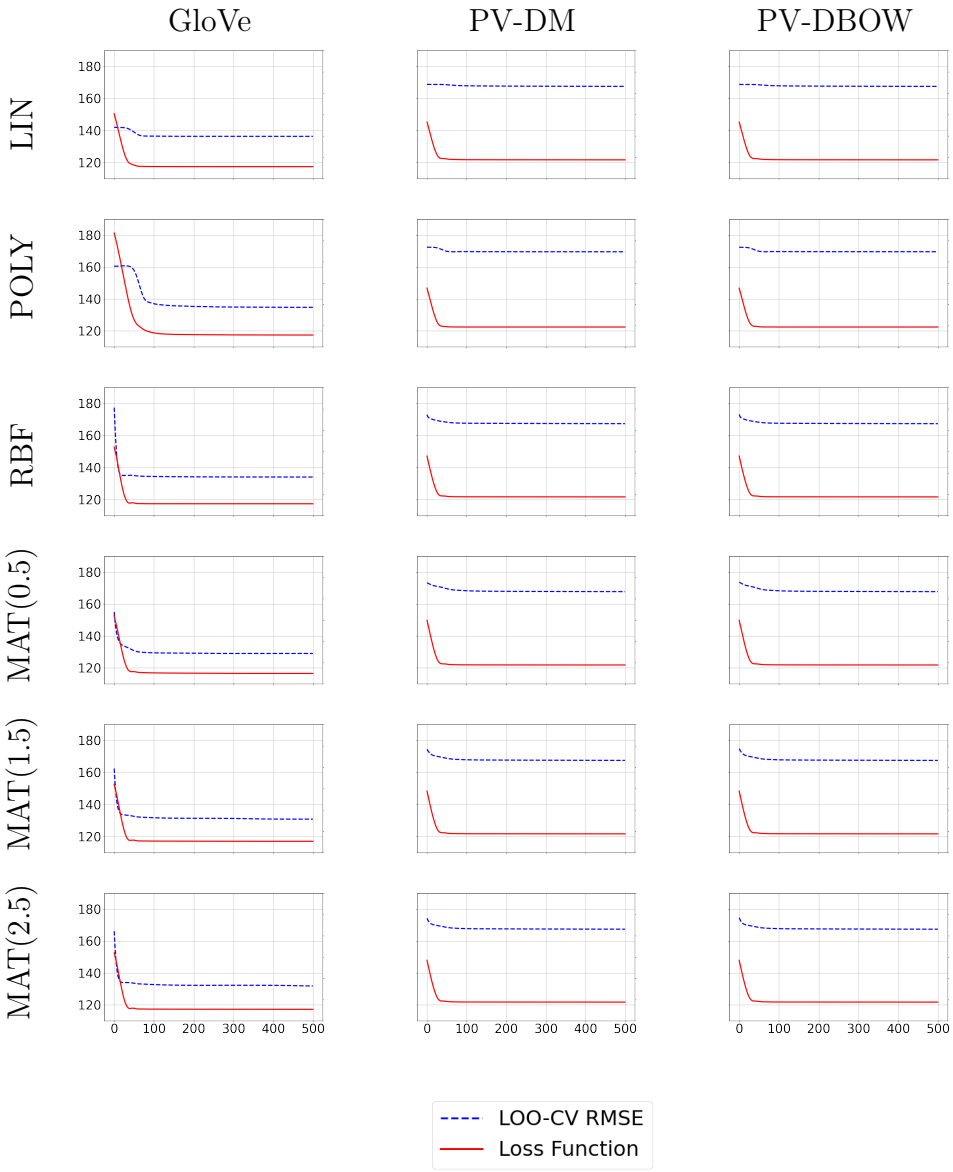


Figure C.1: Training iterations $\tau$ versus the LOO-CV RMSE metrics and loss functions. The tick labels for the loss function were omitted as we are only concerned with observing the number of iterations needed for it to converge.